Universidad
Carlos III de Madrid

Departamento de Ingeniería de Sistemas y Automática

INGENIERÍA INDUSTRIAL
ESPECIALIDAD TECNOLOGÍAS ENERGÉTICAS

PROYECTO FIN DE CARRERA

# INTERFAZ DE CONTROL CINEMÁTICO PARA EL HUMANOIDE HOAP-3 EN MATLAB PARA EL CÁLCULO OFF-LINE DE TRAYECTORIAS

Autor: Daniel Juan García-Cano Locatelli

Tutor: Santiago Martínez de la Casa

Leganés, Marzo de 2012

Título: Interfaz de control cinemático para el humanoide HOAP-3 en Matlab para el cálculo off-line de trayectorias
Autor: Daniel Juan García-Cano Locatelli
Tutor: Santiago Martínez de la Casa

EL TRIBUNAL

Presidente: Alberto Jardón Huete

Vocal: Jorge Arrieta

Secretario: Miguel González-Fierro Palacios

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 7 de Marzo de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

# Agradecimientos

En primer lugar, me gustaría agradecer a mis padres, José y Daniela, y a mi hermano Pablo, todo el apoyo y el cariño que siempre me dan y decirles que si hoy estoy aquí es gracias a ellos.

A Paolo Pierro, que fue mi tutor inicialmente, gracias por la ayuda y el trato que me ha dado desde el primer día, estoy también aquí presentando este proyecto gracias a él.

A mi tutor actual y director del proyecto, también darles las gracias por su ayuda y colaboración.

Han pasado 10 años desde que comencé este camino, Ingeniería Industrial, y si con algo me he de quedar es con los compañeros y compañeras que se han convertido en amigos de verdad con el paso de estos años de alegrías, esfuerzos y penas. A mi compañero y amigo Pablo Salgado, esta carrera nos ha unido como compañeros en una profesión, pero también nos ha unido como amigos auténticos, estando en los buenos y en los malos momentos. A mis compañeros y amigos Kike y Camacho. A mi compañera y amiga Altamira.

A mis amigos César, Pablo, Serguey y Eloy, porque la música nos ha unido, y gracias a ella he podido desconectar en los momentos más duros de esta carrera, y en aquellos momentos difíciles que la vida nos ha dejado en todos estos años.

Por último, me gustaría dedicar este proyecto también a mi abuelo Giovanni Luigi Locatelli, Ingeniero meticuloso, preciso y vocacional, que nos dejó el 20 de febrero de 2012. A mi tía Juana, a mi abuela Carmen, a mi abuelo Evaristo y a mi abuela María Rosa. Los cinco, allá donde estén sé que estarán orgullosos de mí.

# Resumen

En este proyecto se va a desarrollar una interfaz gráfica de usuario para generar trayectorias offline del robot humanoide Hoap-3.

El lenguaje de programación empleado es MATLAB, y la interfaz propuesta es un programa multipantalla con dos principales actividades. Con la primera el usuario podrá acceder al control cinemático de cada manipulador de forma aislada, generando trayectorias en el campo de trabajo y calculando su inversión cinemática posterior. En la segunda, se podrá estudiar la generación de un paso del robot Hoap-3, diseñando los movimientos de los brazos con trayectorias interpoladas al igual que con las piernas.

Los resultados de la interfaz se muestran en gráficas por pantalla, y las estructuras de datos así como las matrices de ángulos generados se pueden almacenar en archivos csv para su posterior uso en simuladores.

El lector podrá ver que se ha creado un modelo en CATIA V5 para poder compaginarlo con la potente herramienta matemática que es MATLAB y comprobar en otra plataforma algunos de los resultados obtenidos.

Así mismo, el presente documento consta de una análisis económico del coste del proyecto y de la hipotética venta del mismo a un cliente empresa.

Para finalizar, el usuario de la interfaz podrá acceder a un manual de usuario así como a las principales funciones creadas para este trabajo, ambas en los anexos.

# Abstract

In this project we are going to develop a graphic user interface (GUI) able to generate offline path time of the humanoid Hoap-3.

The employed program language is MATLAB, and the proposed interface is a multi-display consisting in two principal activities. First of them, user will be able to control the cinematic of every manipulator, generating path time in operational space and working out the inverse cinematic afterwards. During the second activity, we will be able to study the humanoid's step, designing arm's movements with interpolated trajectories as well as with leg's movements.

The results are showing in panels windows, and the datas structures as well as the generated angle's matrix' can be saved into csv files for using them later on in simulators.

The reader can appreciate that it was created a CATIA V5 model with the purpose to be used with the powerfull math tool's that means MATLAB, and to check some of results in other platform.

At the same time, this resume contains the corresponding economical analysis and its hypothetical case of sell.

Finally, the interface user will be able to consult the user's manual and the main functions created for this project, both attached.

# Índice

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción y objetivos

# 1.1 Motivación

La revolución tecnológica acaecida en los últimos 35 años ha supuesto un gran impacto en nuestra vida diaria tal y como se muestra de manera manifiesta en la potencia y velocidad con la que realizamos tareas que antes nos ocupaban más tiempo y recursos.

En este campo, la robótica ha tenido una enorme implicación, automatizando toda clase de tareas y convirtiéndose en uno de los campos de investigación más excitantes.

El aumento de la capacidad computacional de los equipos actuales nos permite simular de manera más eficiente todas las operaciones que los robots y autómatas pueden realizar, por lo que el nuevo reto consiste en replicar todas aquellas tareas que los robots no son aún capaces de emular.

En este sentido, la creación de una interfaz capaz de implementar trayectorias específicas en un robot, puede ser una valiosa herramienta para alcanzar la plena contribución de la automatización en la eliminación de la mano de obra humana para tareas repetitivas o de alta complejidad.

# 1.2 Objetivos

El objetivo fundamental del proyecto es diseñar una interfaz gráfica de usuario en MATLAB para controlar los movimientos del robot off-line y generar las trayectorias de movimiento del mismo.

Además del problema inicial, se van abordar los siguientes objetivos secundarios:

- Introducción a la cinemática directa e inversa del robot humanoide Hoap-3 y comprensión de los espacios articulares y operacionales que se presentan.
- Estudio del algoritmo de resolución de cinemática inversa y diferencial de segundo orden.
- Estudio de las posibles trayectorias y caminos a generar para los manipuladores del robot.
- Estudio de las diversas técnicas de programación para la creación de GUI's (Graphic User Interfaces) en MATLAB.
- Estudio de objetos gráficos en MATLAB empleados para representar los resultados de trayectorias e inversión cinemática.

# 1.3 Fases de desarrollo

El desarrollo general del proyecto ha seguido los siguientes pasos:

1. ***Fase de análisis:*** Planteamiento global del problema y recopilación de información. Qué se quiere resolver y de qué herramientas se dispone para hacerlo, así como el planteamiento de las variables incógnita y los valores iniciales. recogida y criba de información que pueda ser de utilidad para analizar la cuestión más en profundidad.
2. ***Fase de diseño:*** proposición de un diseño inicial de la interfaz usuario. Planteamiento de los principales manipuladores a controlar y posibles tipos de trayectorias a generar y simular.
3. ***Fase de desarrollo:*** Modificación de los algoritmos de resolución de inversión cinemática y desarrollo del código de la interfaz. Programación y compilación de estos algoritmos adaptados a la generación de un paso y a solución aislada de cada manipulador.
4. ***Fase de evaluación:*** Resolución y depuración de resultados***.*** Obtención de resultados en una primera versión del software y depuración del código generado para la obtención de los objetivos propuestos. Validación del diseño de la interfaz planteada.
5. ***Fase de documentación:*** desarrollo de la memoria del proyecto y la documentación relacionada.

# 1.4 Medios empleados

Para la realización de este informe se han empleado los siguientes medios:

- ***Apuntes:*** una parte importante de la información necesaria la hemos obtenido de la abundante colección de apuntes de asignaturas de la Universidad Carlos III de Madrid que versan sobre Automatización y Métodos Numéricos.
- ***Libros:*** sobre las mismas materias detalladas en el primer punto.
- ***Hardware:*** robot HOAP-3 de Fujitsu, ordenador MacBookPro de Apple, y portátil Dell, equipos informáticos de la UC3M, calculadora, impresora y escáner.
- ***Software:*** Mac OS X, MATLAB R2011a, paquete ofimático Office 2008 (Excel, Word y Power Point) para Mac OS X, Windows Xp, Photoshop e Illustrator para el retoque de algunas figuras, Adobe Reader para elaboración de documentos en formato pdf.

- ***Otros medios:*** documentos y archivos proporcionados por el Departamento de Ingeniería de Sistemas y Automática tales como manuales, libros, etc.

# 1.5 Estructura de la memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo:

- **Capítulo 1. Introducción y objetivos:**

Motivación del proyecto, objetivos, fases y medios empleados.

- **Capítulo 2. Estado del arte. Robot HOAP-3**

Información acerca de la tecnología empleada, más concretamente, el robot HOAP-3 de Fujitsu facilitado por la UC3M.

- **Capítulo 3. Análisis matemático:**

Se detallan las principales herramientas matemáticas empleadas, las bases de la cinemática inversa y los diferentes tipos de trayectorias que genera la interfaz.

- **Capítulo 4. Diseño de la interfaz:**

Estructura del programa y principales funciones diseñadas para la interfaz de usuario.

- **Capítulo 5. Funcionamiento de la interfaz:**

Explicación de las principales instrucciones para manejar el software y las diferentes tareas que se pueden realizar.

- **Capítulo 6. Estudio económico del proyecto:**

Planteamiento de la organización del proyecto y las bases presupuestales para generar un supuesto balance económico del mismo.

- **Capítulo 7. Futuras líneas de investigación:**

Propuesta de futuros trabajos y formas de afrontarlos como proyectos encaminados en la misma dirección que el presente.

- **Capítulo 8. Conclusiones y resultados:**

Análisis de la puesta en marcha de la interfaz de usuario y balance de los resultados comparados con los objetivos iniciales del proyecto.

- **Capítulo 9. Bibliografía:**

Libros, artículos de investigación, proyectos fin de carrera y tesis de máster.

- **Anexo 1. Código Matlab:**

Código Matlab de las principales funciones creadas para la interfaz.

- **Anexo 2. Manual de usuario de la interfaz**

Documento de ayuda para el usuario de la interfaz gráfica, detallando las opciones de cada menú.

# Capítulo 2

# Estado del arte. Robot HOAP-3

# 2.1 Estado del arte

La robótica humanoide es un claro ejemplo de la rápida evolución de la robótica. En este momento, los humanoides son capaces de caminar en línea recta, laterales, hacia arriba y abajo, a sentarse y levantar, patear una pelota, de seguir un objetivo, hacer gestos, a bailar, a tocar la trompeta, el piano y otros instrumentos, para interactuar con humanos, y así sucesivamente. Estas plataformas de humanoides demuestran la eficacia de las actividades de investigación, pero todavía cuestan demasiado para convertirse en productos de mercado y, sobre todo, por el momento solo se reproducen unas pocas capacidades de su modelo natural, el ser humano.

En los próximos 10-20 años probablemente se produzca un avance significativo en estas capacidades. Los humanoides podrán contar con las principales características que distinguen al ser humano de los otros animales: la mano y las capacidades del cerebro. Los humanoides actuales demuestran que la tecnología ya ha alcanzado un buen nivel de calidad, especialmente en términos de componentes de transmisión y actuación. Los nuevos paradigmas de diseño tienen que ser utilizados para el desarrollo de las futuras generaciones. Algunos grupos de investigación ya llevan tiempo trabajando con la colaboración de neurocientíficos para ir, literalmente, más allá de la robótica. El aprendizaje humano, cognitivo, de predicción, los modelos perceptivos neurofisiológicos van a ser "traducido" en los nuevos modelos de la robótica. En los próximos 10-20 años, los humanoides integrarán estos nuevos componentes y los resultados se esperan increíbles. El humanoide del futuro será un compañero real, podrá ayudar al ser humano en las actividades de la vida diaria e incluso sustituirle en alguna de ellas. Será capaz de cocinar platos tradicionales, limpiar la casa, hacer la compra, ayudar a las personas mayores, y así sucesivamente. Estos humanoides se caracterizarán por una sofisticada inteligencia artificial y, probablemente, una especie de conciencia que se genere automáticamente.

Las cuestiones éticas a la cuales se enfrenta el ser humano se empiezan a tener en cuenta en equipos multidisciplinares de investigadores, científicos, teólogos, políticos y representantes institucionales con el fin de definir una estrategia común. ¿Seremos capaces de evitar el concepto "Blade Runner" de humanoide, considerarlo un amigo y no un esclavo?

Mientras tanto, en el campo de la programación, hoy en día se están poniendo todas las herramientas disponibles al alcance de cualquier usuario con mínimos conocimientos de programación para la creación de entornos interactivos máquina-hombre: las interfaces gráficas de usuario.

Las últimas versiones de MATLAB van incorporando cada vez más, funciones y objetos realizados en lenguaje java que permiten al usuario del programa un control

más potente orientado a la programación de objetos. De ahí que la aplicación que tiene MATLAB para la creación de GUI's, GUIDE, sea cada día más intuitiva y eficaz, permitiendo en menos tiempo y con menos esfuerzo, diseñar interfaces complejas que antes requerían largas líneas de código.

A esto hay que añadir la interacción de MATLAB con otras plataformas y leguajes como C++, VBASIC, JAVA, etc. La potente pero aún mejorable herramienta Virtual Reality Toolbox, podrá permitir complejos entornos de simulación en breve tiempo.

# 2.2 El robot HOAP-3 de Fujitsu

Fujitsu Automation en colaboración con Fujitsu Lab han desarrollado el robot humanoide en miniatura HOAP (Humanoid for Open Architecture Platform).

La primera versión de este Robot HOAP-1, que salió a mercado en el año 2001, seguida por la segunda versión HOAP-2 en el año 2003, han dado lugar al desarrollo de la tercera versión de dicho Robot, HOAP-3, en el año 2005. La siguiente figura muestra el aspecto del Robot HOAP-3:


Figura 2.2.1 Robot HOAP-3 de Fujitsu

Desde la venta de la primera versión HOAP-1 en 2001, la serie de robots humanoides HOAP ha sido cada vez más utilizada en los departamentos de ingeniería mecánica de las universidades, así como en las áreas de investigación robótica.

## 2.2.1 Estructura del robot

El robot HOAP-3 posee 28 grados de libertad (GDL) distribuidos como se muestra en la figura 2.2.1.1, con sus uniones y extremidades. En lo respectivo a sus dimensiones, tiene una altura de 60 cm y un peso aproximado de 8 Kg.

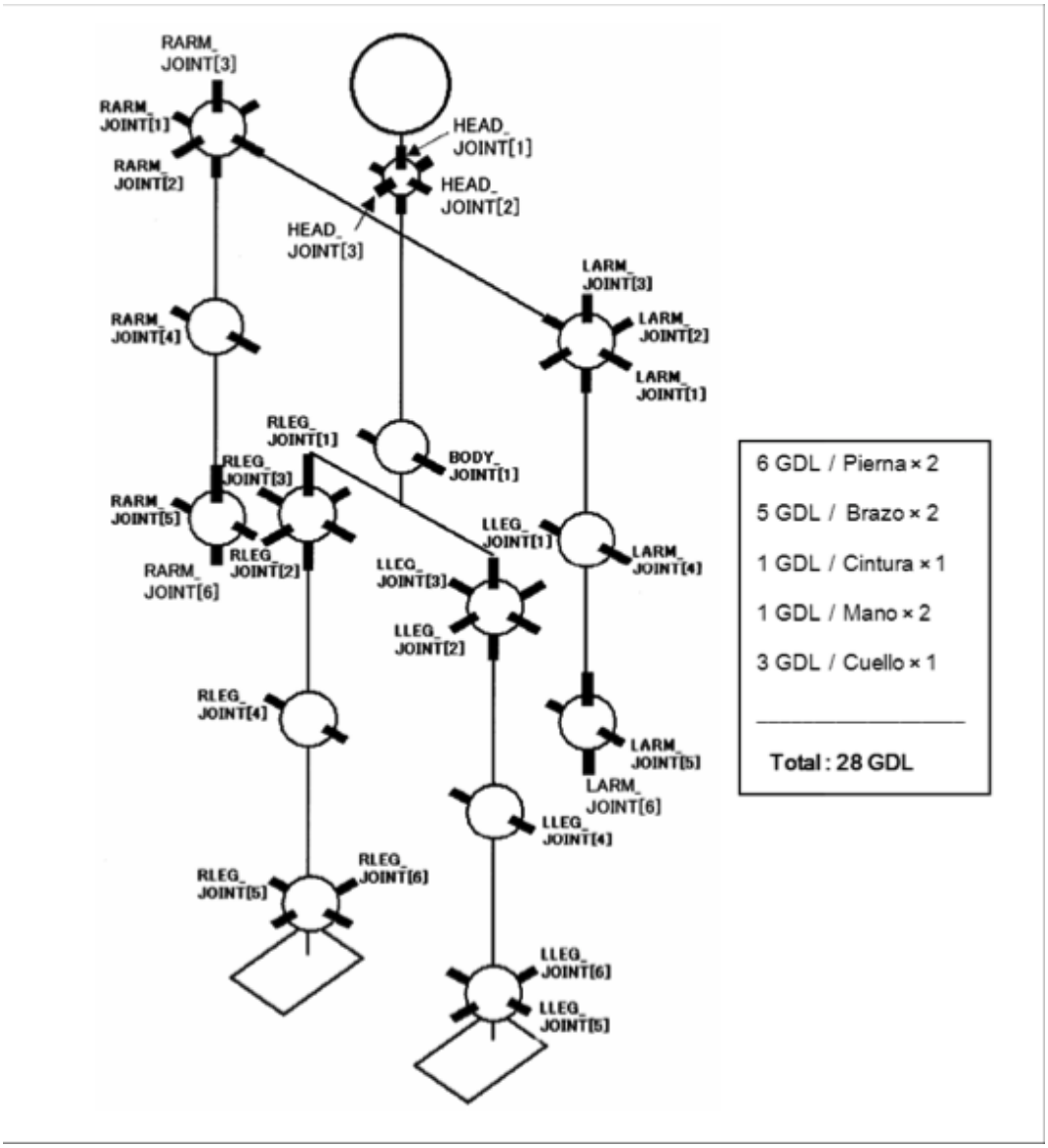



Figura 2.2.1.1 Configuración de las uniones y grados de libertad del robot HOAP-3

Se va a usar como notación para los ejes principales, ZYX, y se llamará a sus correspondientes rotaciones Yaw-Pitch-Roll.

En la tabla 2.2.1.1 se recogen los movimientos asociados a cada una de las articulaciones en función de los ejes principales:

| Articulación | Identificador | Movimiento |
|---|---|---|
| RLEG_JOINT1 | 1 | Right hip joint torsion |
| RLEG_JOINT2 | 2 | Right hip joint roll |
| RLEG_JOINT3 | 3 | Right hip joint pitch |
| RLEG_JOINT4 | 4 | Right knee |
| RLEG_JOINT5 | 5 | Right ankle pitch |
| RLEG_JOINT6 | 6 | Right ankle roll |
| RARM_JOINT1 | 7 | Right shoulder pitch |
| RARM_JOINT2 | 8 | Right shoulder roll |
| RARM_JOINT3 | 9 | Right shoulder torsion |
| RARM_JOINT4 | 10 | Right elbow |
| LLEG_JOINT1 | 11 | Left hip joint torsion |
| LLEG_JOINT2 | 12 | Left hip joint roll |
| LLEG_JOINT3 | 13 | Left hip joint pitch |
| LLEG_JOINT4 | 14 | Left knee |
| LLEG_JOINT5 | 15 | Left ankle pitch |
| LLEG_JOINT6 | 16 | Left ankle roll |
| LARM_JOINT1 | 17 | Left shoulder pitch |
| LARM_JOINT2 | 18 | Left shoulder roll |
| LARM_JOINT3 | 19 | Left shoulder torsion |
| LARM_JOINT4 | 20 | Left elbow |
| BODY_JOINT1 | 21 | Waist pitch |
| HEAD_JOINT1 | 22 | Head torsion |
| HEAD_JOINT2 | 22 | Head pitch |
| HEAD_JOINT3 | 22 | Head roll |
| LARM_JOINT5 | 23 | Left fingers open/close |
| RARM_JOINT5 | 23 | Right fingers open/close |
| LARM_JOINT6 | 23 | Left hand torsion |
| RARM_JOINT6 | 23 | Right hand torsion |

Tabla 2.2.1.1 Notación de los ovimientos de las articulaciones del robot HOAP-3

En el modelo no se han incluido los grados de libertad correspondientes a las manos, articulaciones LARM_JOINT5 y RARM_JOINT5. Estas sirven para abrir y cerrar los dedos permitiendo el agarre de objetos.

No obstante, cabe destacar que no es posible asignar valores a las articulaciones CHEST_Y, CHEST_P, CHEST_R, RARM_WRIST_Y y LARM_WRIST_Y. Estas articulaciones permanecerán en reposo durante toda la trayectoria.

En la siguiente figura se muestra el criterio de signos según los parámetros de Denavit-Hartenberg. Algunos motores están dispuestos en sentido contrario a este criterio, pero este hecho ya ha sido tenido en cuenta al diseñar la interfaz. Por tanto, para crear trayectorias por medio de esta aplicación hay que ceñirse a los criterios de Denavit- Hartenberg en todos los casos.

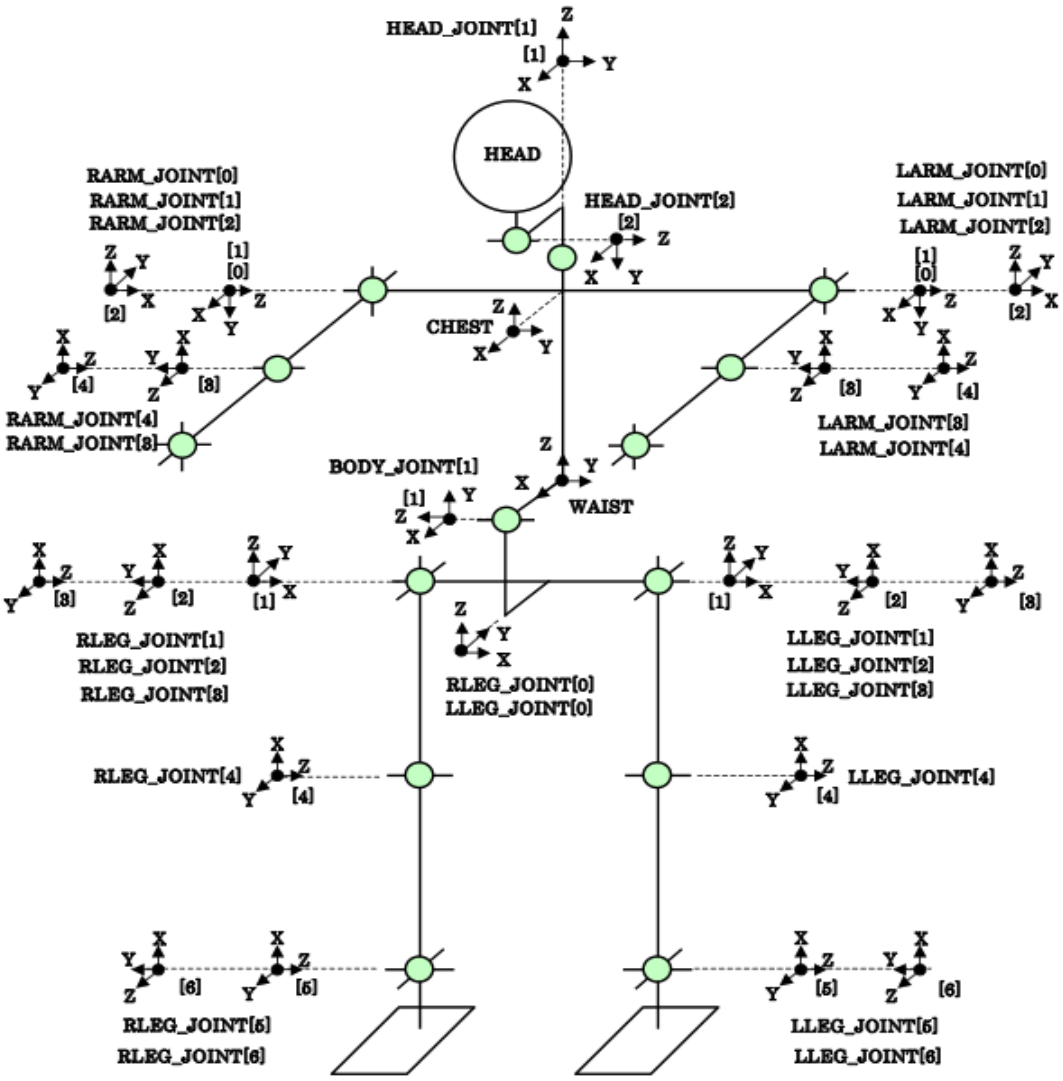



Figura 2.2.1.2 Criterio de signos de los parámetros de Denavit-Hartenberg

La posición de reposo para el robot, en la que todos los encoders de los motores marcan 0 grados, corresponde a la posición de la figura anterior, en que el robot tiene los brazos extendidos en perpendicular al tronco.

## 2.2.2 Dimensiones del robot

En la siguiente tabla se recogen las dimensiones del robot entre las distintas posiciones de las articulaciones, además, se adjunta en la figura 2.2.2.1 la situación de dichas dimensiones así como los grados de libertad representados por líneas de cada una de esas extremidades.

| | | |
|---|---|---|
| ARM_LINK1 | 0.111 | m |
| ARM_LINK2 | 0.111 | m |
| ARM_LINK3 | 0.171 | m |
| LEG_LINK1 | 0.039 | m |
| LEG_LINK2 | 0.105 | m |
| LEG_LINK3 | 0.105 | m |
| LEG_LINK4 | 0.040 | m |

| | | |
|---|---|---|
| BODY_LINK1 | 0.125 | m |
| BODY_LINK2 | 0.035 | m |
| HEAD_LINK1 | 0.103 | m |
| HEAD_LINK2 | 0.015 | m |
| WAIST_LINK1 | 0.055 | m |
| WAIST_LINK2 | 0.035 | m |

Tabla 2.2.2.1 Notación de los ovimientos de las articulaciones del robot HOAP-3

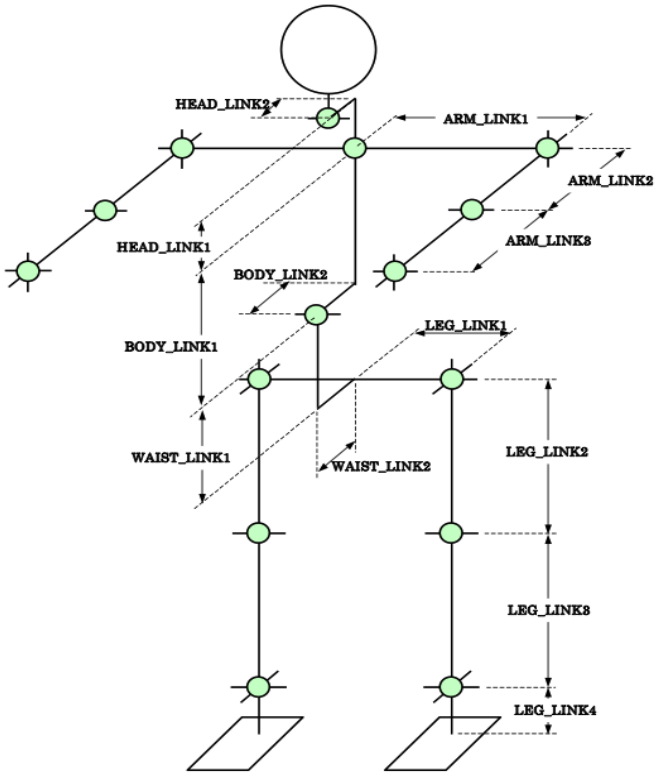



Figura 2.2.2.1 Criterio de signos de los parámetros de Denavit-Hartenberg

## 2.2.3 Dispositivos y sensores del robot

El robot HOAP-3 tiene varios sensores, entre otros de aceleración, de fuerza, o un sensor para la medición de distancias a través de infrarrojos.

Además, está dotado de una cámara, un micrófono, un altavoz y dos LED en la cabeza. La versión 3 incluye características tales como la de mostrar emociones mediante un LED, reconocer imágenes y sonidos, sintetizar la voz y la posibilidad de comunicarse con un humano en un intento de diálogo natural fluido.

La información de Interfaz de hardware y software es de código abierto, por lo que puede ser programado libremente. Todo ello funciona sobre RTLinux y se proporcionan toda clase de herramientas para este sistema operativo con el fin de poder programar el robot y añadir funcionalidades adicionales que dotan al usuario de un control total.

En la siguiente figura se muestra la localización de los principales sensores que se ha mencionado.

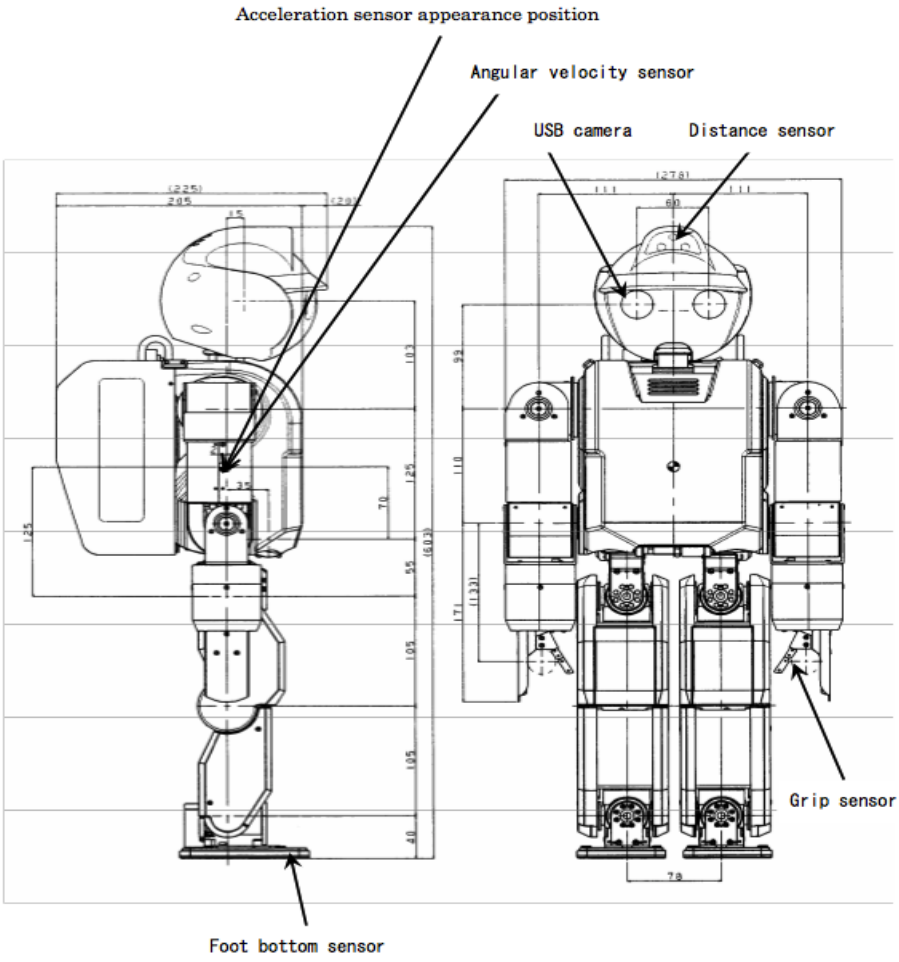



Figura 2.2.3.1 Localización de los sensores del robot HOAP-3

### 2.2.4 Rango posible de movimientos del robot

Con el fin de asegurar los parámetros de seguridad de funcionamiento del robot, se impondrá rangos de valores a los movimientos del robot que coinciden con los topes del encoder de los motores en las articulaciones.

En la práctica, para asegurar un mantenimiento óptimo del equipo (robot), se reduce este rango de movimientos en 1 grado por encima del mínimo y 1 grado por debajo del máximo. Por ejemplo para la articulación LARM_JOINT3 el rango de movimientos quedará entre [-90º, 90º].

En la siguiente tabla, se adjuntan los valores de movimiento mínimo y máximo de cada articulación.

| Articulación | Id | Movimiento asociado | Valor mínimo | Valor máximo |
|---|---|---|---|---|
| RLEG_JOINT1 | 1 | Right hip joint torsion | -91 | 31 |
| RLEG_JOINT2 | 2 | Right hip joint roll | -31 | 21 |
| RLEG_JOINT3 | 3 | Right hip joint pitch | -82 | 71 |
| RLEG_JOINT4 | 4 | Right knee | -1 | 130 |
| RLEG_JOINT5 | 5 | Right ankle pitch | -61 | 61 |
| RLEG_JOINT6 | 6 | Right ankle roll | -25 | 25 |
| RARM_JOINT1 | 7 | Right shoulder pitch | -181 | 61 |
| RARM_JOINT2 | 8 | Right shoulder roll | -96 | 1 |
| RARM_JOINT3 | 9 | Right shoulder torsion | -91 | 91 |
| RARM_JOINT4 | 10 | Right elbow | -115 | 1 |
| LLEG_JOINT1 | 11 | Left hip joint torsion | -31 | 91 |
| LLEG_JOINT2 | 12 | Left hip joint roll | -21 | 31 |
| LLEG_JOINT3 | 13 | Left hip joint pitch | -82 | 71 |
| LLEG_JOINT4 | 14 | Left knee | -1 | 130 |
| LLEG_JOINT5 | 15 | Left ankle pitch | -61 | 61 |
| LLEG_JOINT6 | 16 | Left ankle roll | -25 | 25 |
| LARM_JOINT1 | 17 | Left shoulder pitch | -181 | 61 |
| LARM_JOINT2 | 18 | Left shoulder roll | -1 | 96 |
| LARM_JOINT3 | 19 | Left shoulder torsion | -91 | 91 |
| LARM_JOINT4 | 20 | Left elbow | -115 | 1 |
| BODY_JOINT1 | 21 | Waist pitch | -90 | 1 |

Tabla 2.2.4.1 Rangos de movimiento de las extremidades del robot HOAP-3

# Capítulo 3

# Análisis matemático

# 3.1 Introducción al estudio cinemático del robot

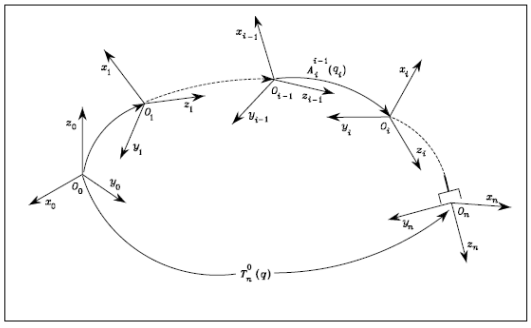
Desde el punto de vista mecánico, los robots son sistemas de cuerpos rígidos conectados por articulaciones. Dado que los robots están diseñados para realizar movimientos, la cinemática es uno de los aspectos fundamentales en el diseño, análisis, control y simulación de robots.

La cinemática estudia el movimiento de estos cuerpos sin tener en cuenta las fuerzas o momentos que los generan. A la posición y orientación de un cuerpo en el espacio se las denomina conjuntamente localización. En particular, la cinemática describe la localización, velocidad y aceleración de un cuerpo que incluye un mecanismo.

A la hora de traducir un objeto tan complejo en un ente matemático, se considera al robot como una cadena cinemática formada por objetos rígidos o eslabones (links) unidos entre sí mediante articulaciones (joints). Un extremo de la cadena se considera el centro de referencia o base y el otro extremo *end-effector*. Para manipular este objeto en el espacio se describe la posición y orientación de este vector extremo final.

Entre las muchas topologías en las que pueden ser conectados los sistemas de eslabones rígidos existen dos formas en particular en robótica: cadenas cinemáticas abiertas y cadenas cinemáticas cerradas.

Una cadena cinemática abierta (Figura 3.1.a) es un sistema de cuerpos rígidos en la cual cada miembro está conectado con otros dos, excepto el primer y último eslabón.



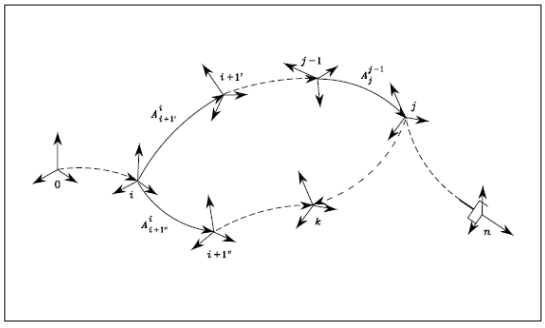


Figura 3.1.a Cadena cinemática abierta  Figura 3.1.b Cadena cinemática cerrada

Una cadena cinemática cerrada (Figura 3.1.b) es aquella en la cual dos cuerpos están conectados por varias articulaciones, y en la práctica, cada articulación es en sí una cadena cinemática abierta.

A la hora de plantear el problema cinemático del robot, se tendrá:

Cinemática directa: A partir de valores conocidos de las articulaciones y parámetros geométricos de los elementos del robot, se calcula la posición y orientación del *end-effector* con respecto a un sistema de referencia.

Cinemática inversa: En un intervalo de tiempo se tiene la posición y orientación del extremo de un objeto o manipulador y se pretende hallar el ángulo de giro de cada articulación en cada instante de tiempo.

A continuación se presentan las herramientas y conceptos matemáticos fundamentales.

## 3.1.1 Localización de un sólido rígido

Para describir un sólido rígido en el espacio hace falta su posición y su orientación respecto de un sistema de referencia.

La forma más común de representar la localización de un sólido en el espacio es mediante un sistema de coordenadas de referencia, o simplemente sistema. Un sistema de referencia consiste de un origen, denotado como O y una triada de vectores base, mutuamente ortogonales, que se consideran fijos con respecto del cuerpo (Figura 3.1.1.1). La localización de un cuerpo siempre se expresa en relación con otro cuerpo, por lo tanto, se puede expresar la localización de un sistema con respecto a otro sistema.

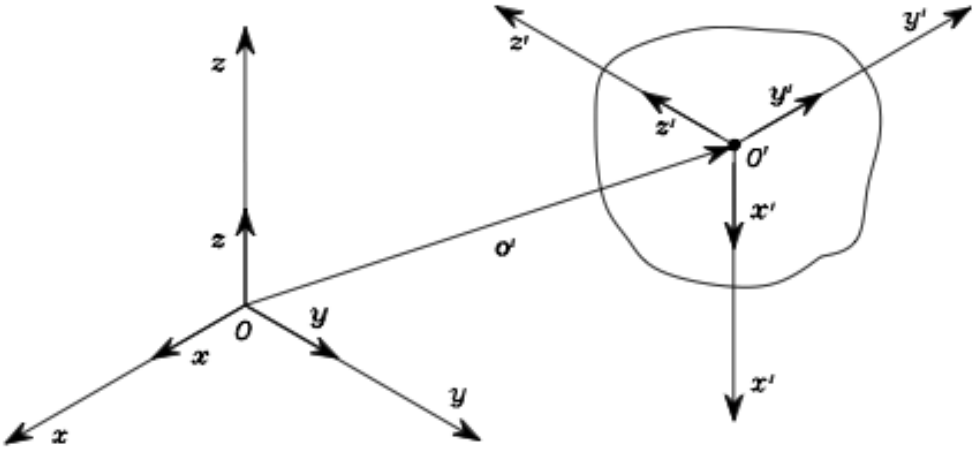



Figura 3.1.1.1 Sistemas de referencia

Durante el proyecto se trabajará con los sistemas coordenados en los cuales los objetos se trasladarán y/o rotarán.

Se considera un sistema inercial *Si* denotado por los ejes coordenados $x_i$,$y_i$ y $z_i$. Este sistema coordenados permanece fijo durante el movimiento de un cuerpo $m_i$ que se encuentra fijo a un sistema coordenado $S_b$, cuyos ejes coordenados son $x_b$,$y_b$ y $z_b$.

En la siguiente figura se representa lo anteriormente expuesto.

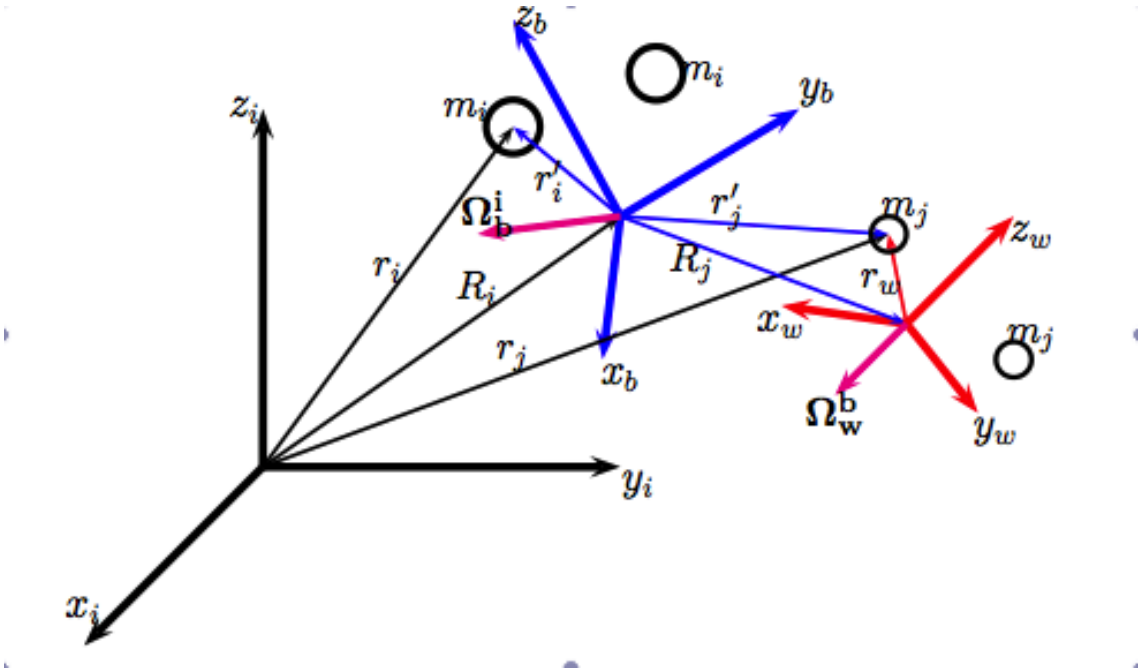



Figura 3.1.1.2 Sistemas de ejes coordenados

Existe también un cuerpo $m_j$ que se mueve en el sistema $S_b$ y este cuerpo está fijo a un tercer sistema llamado $S_w$ definido por los ejes $x_w$,$y_w$ y $z_w$.
Se puede observar que los cuerpos $m_i$ en el sistema Sb tienen un movimiento traslacional $R_i$ y un movimiento rotacional $\Omega_b^i$ respecto al sistema Si. Para visualizar el correcto movimiento de los cuerpos $m_i$ se debe de ajustar la distancia $r_i'$. De la misma manera para el movimiento de los $m_j$ del sistema Sw definido por su traslación $R_j$ y su rotación Delta $b_w$ respecto al sistema $S_b$, se debe definir la distancia $r_w$.

Esto indica que el "observador" se encuentra en una posición estacionaria dentro del sistema fijo y que no existe un sistema de referencia absoluto.

Para definir la posición es común utilizar coordenadas cartesianas (*x,y,z*) y para definir la rotación se puede utilizar los ángulos de Euler, que de entre sus muchas notaciones, para este proyecto se adoptará la convención *roll*($\theta$), *pitch*($\phi$) y *yaw*($\psi$) , tal como muestra la figura 3.1.1.3.

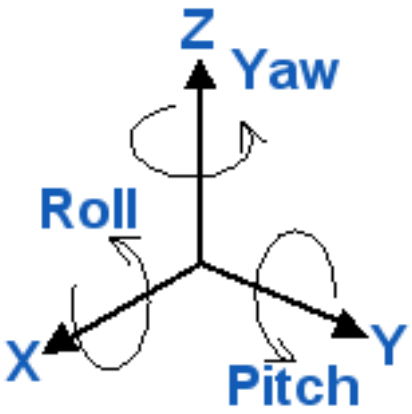



Figura 3.1.1.3 Notación de los giros en coordenadas cartesianas

También se trabajará con cuaternios, que más adelante se explicará, para evitar singularidades que se puedan producir con los ángulos de Euler en el transcurso de los algoritmos.

Los conceptos de espacio de trabajo u operacional y espacio articular se deben tener en cuenta de aquí en adelante.

El espacio de trabajo es el espacio físico donde nos movemos y estamos familiarizados, es la zona que puede ser alcanzada por un punto del extremo final.

El espacio articular es el espacio matemático formado por las variables articulares. De dimensión nx1, donde n es número de grados de libertad que tenga el robot. Cabe destacar que todas las articulaciones que tenemos son giratorias y no prismáticas.

## 3.1.2 Grados de libertad del robot

Los grados de libertad del sistema que se describe (GDL), corresponden al número mínimo de variables independientes que es necesario definir para determinar la situación en el espacio de los eslabones del manipulador.

La regla de K. Kutzbach permite determinar el número de grados de libertad de un robot con mecanismos especiales en términos del número de eslabones y de articulaciones.

Un robot necesita seis grados de libertad para lograr una posición y orientación determinadas de su *end-effector* o extremo del manipulador. Si existen menos de seis grados de libertad, la serie de posiciones y orientaciones alcanzables es limitada, en cambio, si hay más de seis grados de libertad habrá redundancia y por lo tanto existirá un infinito número de posibilidades para alcanzar una posición y orientación del *end-effector*, siendo esto útil cuando si se desea evitar obstáculos u optimizar el tiempo, eligiendo configuraciones alternativas para las articulaciones.

## 3.1.3 Representación de la orientación de un cuerpo

Existen diferentes formas de representar la orientación de un cuerpo, las más comunes son los ángulos de Euler, los parámetros de Euler, las matrices de rotación y los cuaternios unitarios.

Estas herramientas de representación matemática se utilizan en aplicaciones de aeronáutica, aeroespacial, robótica, en simulaciones con modelos de realidad virtual, etc.

**Ángulos de Euler:**

Representan la orientación de un cuerpo respecto a un sistema coordenado xyz o la orientación de un sistema coordenado respecto a otro, como se mostró anteriormente.

Para describir la orientación de un cuerpo en términos de los ángulos de Euler se definen las siguientes rotaciones:

$\theta$ (*roll*) describe la rotación alrededor del eje $x$.

$\phi$ (*pitch*) describe la rotación alrededor del eje $y$.

$\psi$ (*yaw*) describe la rotación alrededor del eje $z$.

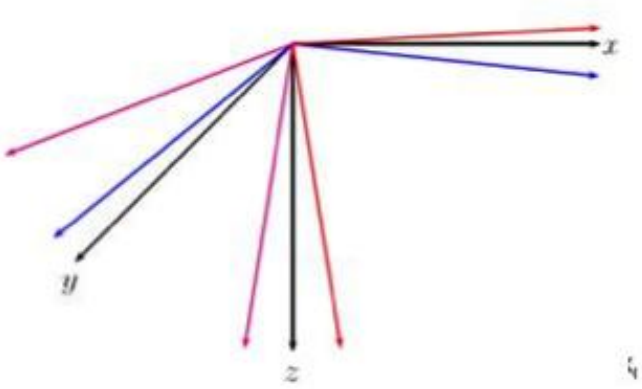



Figura 3.1.3.1 Ángulos de Euler

Esta representación es fácil de visualizar geométricamente, además son usados para definir transformaciones de coordenadas mediante una secuencia de rotaciones.

**Cuaternios unitarios:**

La orientación de un objeto puede ser descrita como una rotación alrededor de un eje. Este eje es un vector unitario con componentes en un sistema coordenados *xyz*, por lo que se necesitan cuatro parámetros para describir una orientación respecto a un sistema coordenado de referencia; los tres componentes del vector unitario y un ángulo de rotación teta.

La ventaja de utilizar cuaternios unitarios es que se evitan singularidades, aquellos puntos donde una función tiende a infinito o está mal definida en otro sentido. Estas singularidades se presentan en secuencias de rotaciones expresadas en ángulos de Euler.

Un cuaternio se define por tener una parte escalar y una vectorial, de manera que:

$$q = \begin{bmatrix} \eta & \epsilon_1 & \epsilon_2 & \epsilon_3 \end{bmatrix}$$

y para un cuaternión unitario, se tiene:

$$\eta^2 + \epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 = q^\top q = 1$$

donde:

$$\eta = \cos\frac{\beta}{2}$$

$$\epsilon = \begin{bmatrix} \epsilon_1 & \epsilon_2 & \epsilon_3 \end{bmatrix} = \lambda \sin\frac{\beta}{2}$$

Se puede ver que ambas partes se expresan en términos de una rotación ß alrededor de un vector unitario gamma. Esto nos servirá para representar la orientación de un cuerpo, conocida como representación en parámetros de Euler.

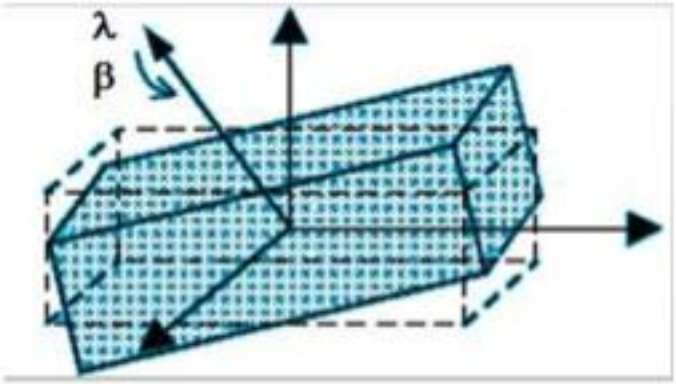



Figura 3.1.3.2 Representación de los parámetros de Euler

## 3.1.4 Transformación de coordenadas

**Matriz de rotación:**

La matriz de rotación transforma un vector en coordenadas de un sistema i a un vector expresado en coordenadas de un sistema j. También provee una representación de la orientación del sistema i con respecto al sistema j, y por lo tanto, puede ser una representación de rotación del sistema i al sistema j.

Una rotación elemental del sistema i alrededor del eje Z un ángulo θ está definida por la matriz:

$$R(z,\theta) = \begin{pmatrix} \cos(\theta) & -\operatorname{sen}(\theta) & 0 \\ \operatorname{sen}(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

El mismo ángulo θ rotado en el eje Y se define por la matriz:

$$R(y,\theta) = \begin{pmatrix} \cos(\theta) & 0 & \operatorname{sen}(\theta) \\ 0 & 1 & 0 \\ -\operatorname{sen}(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

y alrededor del eje X la matriz es:

$$R(x,\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\operatorname{sen}(\theta) \\ 0 & \operatorname{sen}(\theta) & \cos(\theta) \end{pmatrix}$$

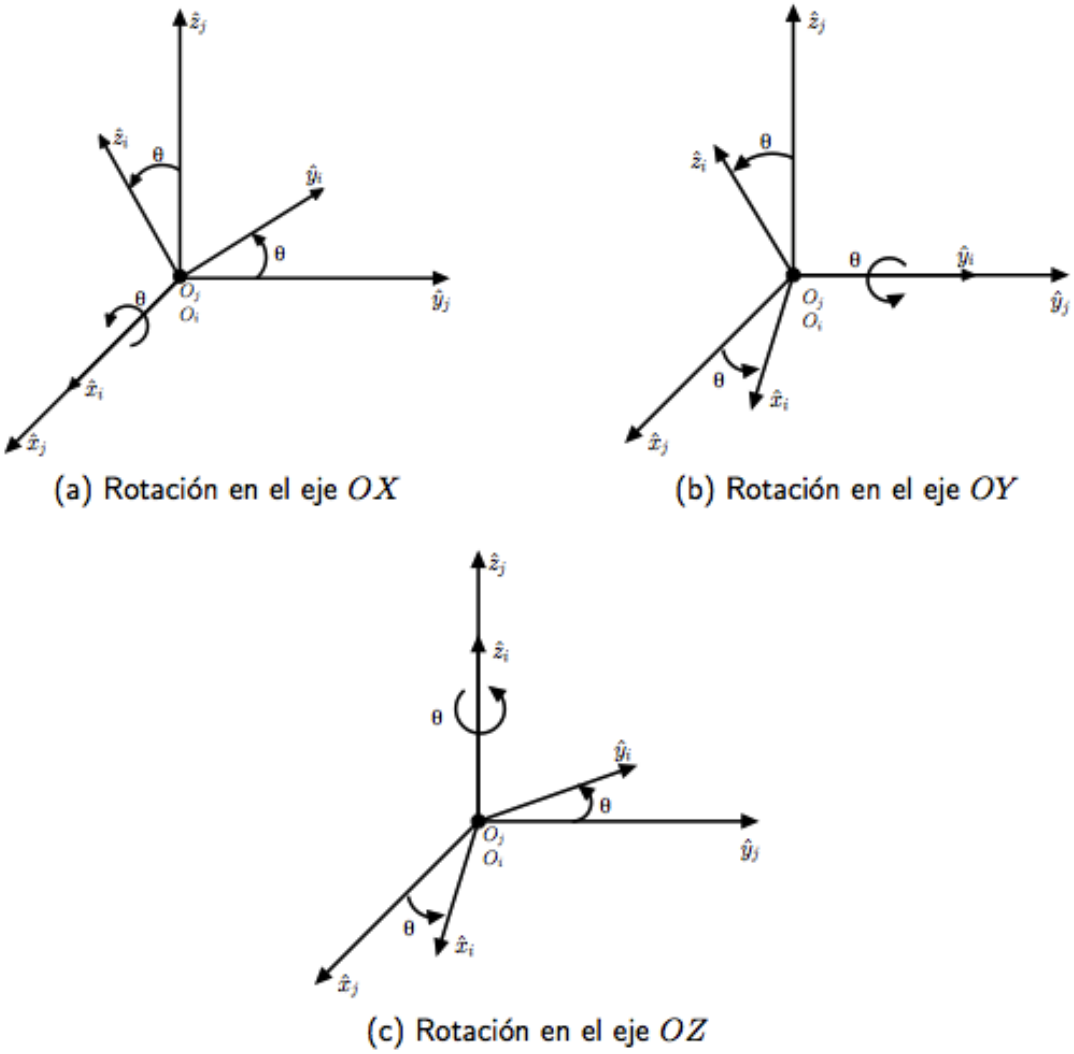



Figura 3.1.4.1 Rotaciones en los ejes coordenados

**Transformaciones homogéneas:**

Las estructuras matemáticas presentadas anteriormente sólo representan la posición o la orientación. Con las transformaciones homogéneas, los vectores de posición y las matrices de rotación son combinados en una sola estructura matemática compacta. Por ejemplo:

La matriz de transformación homogénea de una simple rotación alrededor de un eje, en ocasiones es denotada como *Rot* de tal manera que una rotación de un ángulo θ alrededor del eje zˆ está dada por la matriz:

$$\mathbf{Rot}(\hat{z},\theta) = \begin{pmatrix} \cos(\theta) & -\operatorname{sen}(\theta) & 0 & 0 \\ \operatorname{sen}(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

De igual manera, la matriz de transformación homogénea de una simple traslación a lo largo de un eje se denota como *Trans*. De modo que una traslación de una distancia d a lo largo del eje X es:

$$\mathbf{Trans}(\hat{x},d) = \begin{pmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Las matrices de transformación homogénea son atractivas cuando se desea una notación compacta y/o una programación sencilla. Sin embargo su costo computacional no es muy eficiente, dado que la composición de estas matrices requiere de varias multiplicaciones.

Matriz Jacobiana: establece la relación entre las velocidades de las articulaciones y el órgano final o end effector.

$$\mathbf{J(\theta)} = \left( \frac{\partial x_i}{\partial \theta_j} \right)_{i,j}$$

**Ecuación 1**

donde, la dimensión de la matriz es ***m*** x ***n***, donde el índice ***m*** representa la cantidad de funciones escalares ( *x*, *y*, *z*,α , β ,γ) y el índice ***n*** representa el número de grados de libertad de una determinada cadena cinemática.

# 3.2 Cinemática directa

La cinemática directa consiste en encontrar la localización relativa de dos eslabones en particular, de acuerdo a la geometría del robot y los valores de sus n articulaciones.

El problema cinemático directo es crítico para desarrollar algoritmos de coordinación del manipulador del robot porque los valores de las articulaciones son medidos por los sensores montados en las articulaciones, y es necesario calcular los valores de estas articulaciones con respecto a los sistemas asignados.

Se puede establecer un sistema de referencia fijo situado en la base del robot y describir la localización de cada uno de los eslabones con respecto a dicho sistema de referencia. De esta forma, el problema cinemático directo se reduce a encontrar una matriz homogénea de transformación **T** que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base del mismo. Esta matriz **T** será función de las coordenadas articulares.

También podría encontrarse la solución a este problema a través de relaciones geométricas, pero éste es un método muy poco versátil, que depende de la configuración del robot y sirve para robots de pocos grados de libertad.

Se plantea el problema considerando el caso general de espacio operacional constituido por seis dimensiones (m=6), tres para la posición (x,y,z) y tres para la orientación (roll,pitch,yaw) mediante las siguientes relaciones:

$$x = f_x(q1, q2, \ldots, q_n) \qquad \alpha = f_\alpha(q1, q2, \ldots, q_n)$$
$$y = f_y(q1, q2, \ldots, q_n) \qquad \beta = f_\beta(q1, q2, \ldots, q_n)$$
$$z = f_z(q1, q2, \ldots, q_n) \qquad \gamma = f_\gamma(q1, q2, \ldots, q_n)$$

donde , $q_1$, $q_2$…., $q_n$ representan los valores de las articulaciones en un determinado instante.

$$x_e = f(q)$$

**Ecuación 2**

La cinemática directa consiste en encontrar esta función vectorial (*mx1*) ***f(·)*** , que por lo general será no-lineal, y que permitirá calcular las variables del espacio de trabajo conociendo las variables del espacio articular del robot. **[1]**

## 3.2.1 Representación de Denavit-Hartenberg

En 1955 Denavit-Hartenberg (DH) propusieron un método sistemático que consiste en ubicar un sistema coordenado solidario $S_i$ a cada eslabón i, para pasar de un eslabón a otro, mediante cuatro transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Estas transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento *i* con el sistema del elemento *i-1*.

Las transformaciones son las siguientes:

1. Rotación alrededor del eje $z_i$-*1* un ángulo $\vartheta_i$.
2. Traslación a lo largo de $z_i$-*1* una distancia $d_i$.
3. Traslación a lo largo de $x_i$ una distancia ai.
4. Rotación alrededor del eje $x_i$, un ángulo $\alpha_i$.

Dado que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que:

$$^{i-1}\mathbf{A}_i = \mathrm{T}(z,\theta_i)\,\mathrm{T}(0,0,d_i)\,\mathrm{T}(a_i,0,0)\,\mathrm{T}(x,\alpha_i)$$

y realizando el producto de matrices:

$$^{i-1}\mathbf{A}_i = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Entonces, únicamente requerimos los cuatro parámetros de DH para obtener la matriz de transformación que relaciona a los sistemas coordenados de dos eslabones consecutivos.

# 3.3 Cinemática inversa y diferencial

Uno de los problemas fundamentales y complejos en robótica es la *resolución del problema cinemático inverso* (**PCI**), ya que no se puede encontrar una metodología de solución aplicable a todos los robots, siendo factores determinantes: el número de grados de libertad, el número de efectores finales, el tipo de cadenas cinemáticas (abierta o cerrada), la redundancia, etc.

El problema cinemático inverso consiste en encontrar el estado de cada una de las articulaciones que conforman un robot, para lograr una posición y orientación del efector final, de manera que, considerando las Ecuaciones 1 y 2, se tiene:

$$q = f^{-1}(q) \cdot x_e$$

**Ecuación 3**

Siendo $f^{-1}$ la función que desea encontrar la cinemática inversa. Como anteriormente se hizo, en el espacio operacional de seis dimensiones se tiene que:

$x_e = \begin{pmatrix} p_e \\ \phi_e \end{pmatrix}$ donde $p_e$ es la posición y $\phi_e$ es la orientación del sistema de referencia del extremo del manipulador, y considerando el concepto del Jacobiano:

$$\left.\begin{aligned} \dot{p}_e &= J_p(q) \cdot \dot{q} \\ w_e &= J_O(q) \cdot \dot{q} \end{aligned}\right\} \quad v_e = \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix} = J(q) \cdot \dot{q}$$

**Ecuación 4**

Donde la matrix ***J*** (*6xn*) es la *Jacobiana Geométrica* del manipulador:

$$J = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$$

**Ecuación 5**

Del mismo modo que se tiene una relación (no-lineal por lo general) en la cinemática directa (Ecuación 2), la cinemática diferencial relaciona el vector velocidades del espacio operacional con el vector velocidades del espacio articular (ecuación 4).

Considérese ahora el mismo problema que se plantea en la Ecuación 3 pero con el vector de velocidades del espacio articular. El objetivo ahora de la cinemática inversa diferencial es hallar la inversa de la Jacobiana:

$$\dot{q} = J^{-1}(q) \cdot v_e$$

**Ecuación 6**

Antes de seguir convendría destacar un aspecto importante de aquí en adelante. La matriz Jacobiana Geométrica, está referenciada respecto del sistema referencia base u origen del manipulador, que suele ser el centro de masas (CoM) del robot. Por ello se trabaja con la velocidad angular *w*, que es la velocidad de giro del sistema de referencia del extremo del manipulador respecto a una base de referencia (CoM). Si se considera la orientación del sistema de referencia $\phi_e$ , su derivada en el tiempo (su velocidad de rotación) difiere de la velocidad angular w. Llegados a este punto se define la Jacobiana Analítica y su relación con la Jacobina Geométrica:

$$\left.\begin{aligned} \dot{p}_e &= \frac{\partial p_e}{\partial q} \cdot \dot{q} \\ \dot{\phi}_e &= \frac{\partial \phi_e}{\partial \phi} \cdot \dot{q} \end{aligned}\right\} \quad \dot{x}_e = \begin{bmatrix} J_P(q) \\ J_\phi(q) \end{bmatrix} \cdot \dot{q} = J_A(q) \cdot \dot{q}$$

**Ecuación 7**

Siendo $J_A$ la matriz Jacobiana Analítica, y la relación con la Jacobiana Geométrica:

$$J = T_A(\phi) \cdot J_A$$

**Ecuación 8**

Siendo TA la matriz de transformación entre $w_e$ y $\phi_e$

Conociendo la configuración inicial del manipulador que se esté considerando, se tiene:

$$q(t) = \int_0^t \dot{q}(\eta) d\eta + q(0)$$

$$q(t_{k+1}) = q(t_k) + \dot{q}(t_k) \cdot \Delta t$$

La solución numérica a esta integral, y considerando la ecuación 6 evaluada en el instante previo, se tiene que:

$$q(t_{k+1}) = q(t_k) + J^{-1}(q(t_k))\, v_e(t_k) \cdot \Delta t$$

**Ecuación 9**

La diferencia entre las ecuaciones 9 y 6 muestra que las variables articulares calculadas difieren de las variables articulares perseguidas o deseadas.

Para afrontar este problema se define el error del espacio operacional entre la localización (posición y orientación) $\boldsymbol{x_d}$ deseada y la localización actual $\boldsymbol{x_e}$ y su derivada en el tiempo:

$$\left.\begin{array}{l} e = x_d - x_e \\ \dot{e} = \dot{x}_d - \dot{x}_e \\ x_e = J_A(q)\cdot \dot{q} \end{array}\right\} \dot{e} = \dot{x}_d - J_A(q)\cdot \dot{q}$$

**Ecuación 10**

Sea por tanto, el algoritmo de inversión cinemática siguiente:

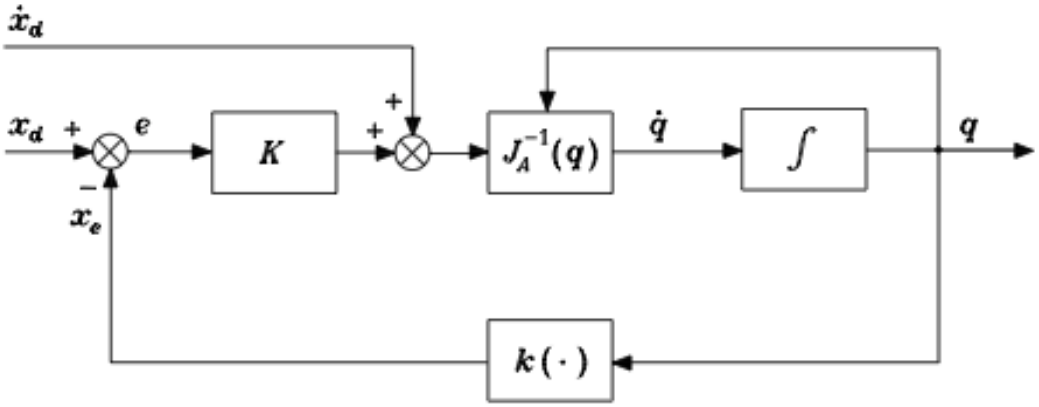



Figura 3.3.1: Algoritmo de cinemática inversa con la jacobiana inversa

Este algoritmo de resolución indica que las variables articulares q de una localización deseada xd, se calculan cuando el error xd – f(q) es muy pequeño (considerando un margen). El tiempo de convergencia dependerá de las características dinámicas del error de la ecuación 10.

Considerando la ecuación 10 y la siguiente ecuación 11:

$$\dot{q} = J_A^{-1}(q)\cdot\left(\dot{x}_d + K\cdot e\right)$$

**Ecuación 11**

Se tiene el siguiente sistema lineal equivalente:

**Ecuación 12**

$$\dot{e} + K\cdot e = 0$$

en la ecuación 12, se definiría K como una matriz (diagonal) para que el sistema sea asintóticamente estable, esto es, el error tiende a cero dependiendo a lo largo de la trayectoria según sean los valores propios o eigenvalues de esta matriz K. **[1]**

Notar, que en la figura 3.3.1, el bloque k(·) representa la cinemática directa que realimenta el bucle calculando la posición actual $x_e$.

El objetivo de este proyecto es trabajar con el algoritmo de segundo orden, por todo esto se ha analizado previamente el anterior algoritmo, considerado de primer orden.
Las razones residen en la dinámica, debido a que se considera a los manipuladores como sistemas mecánicos de segundo orden. A nivel de control, se desea invertir una trayectoria de movimiento específica en términos de la *posición, velocidad y aceleración*.

Para presentar el algoritmo protagonista de la inversión cinemática que se realiza en la interfaz, se tendrán en cuenta las siguientes ecuaciones:

Derivando la ecuación 7 se tiene:

$$\ddot{x}_e = J_A(q)\cdot \ddot{q} + \dot{J}_A\left(q,\dot{q}\right)\cdot \dot{q}$$

**Ecuación 13**

Derivando la ecuación 10 se tiene:

$$\ddot{e} = \ddot{x}_d - \ddot{x}_e$$

**Ecuación 14**

Derivando la ecuación 11 se tiene:

$$\ddot{q} = J_A^{-1}(q)\cdot\left(\ddot{x}_e - \dot{J}_A\left(q,\dot{q}\right)\cdot \dot{q}\right)$$

**Ecuación 15**

Por lo que la aceleración del espacio articular será:

$$\ddot{q} = J_A^{-1}(q)\cdot\left(\ddot{x}_d + K_D\cdot \dot{e} + K_P\cdot e - \dot{J}_A\left(q,\dot{q}\right)\cdot \dot{q}\right)$$

**Ecuación 16**

Y dará lugar al sistema lineal del error siguiente:

$$\ddot{e} + K_D \cdot \dot{e} + K_P \cdot e = 0$$

**Ecuación 17**

Este sistema lineal será asintóticamente estable dependiendo de la elección de las matrices $K_P$ y $K_D$. Véase el esquema del algoritmo de segundo orden en la siguiente figura:

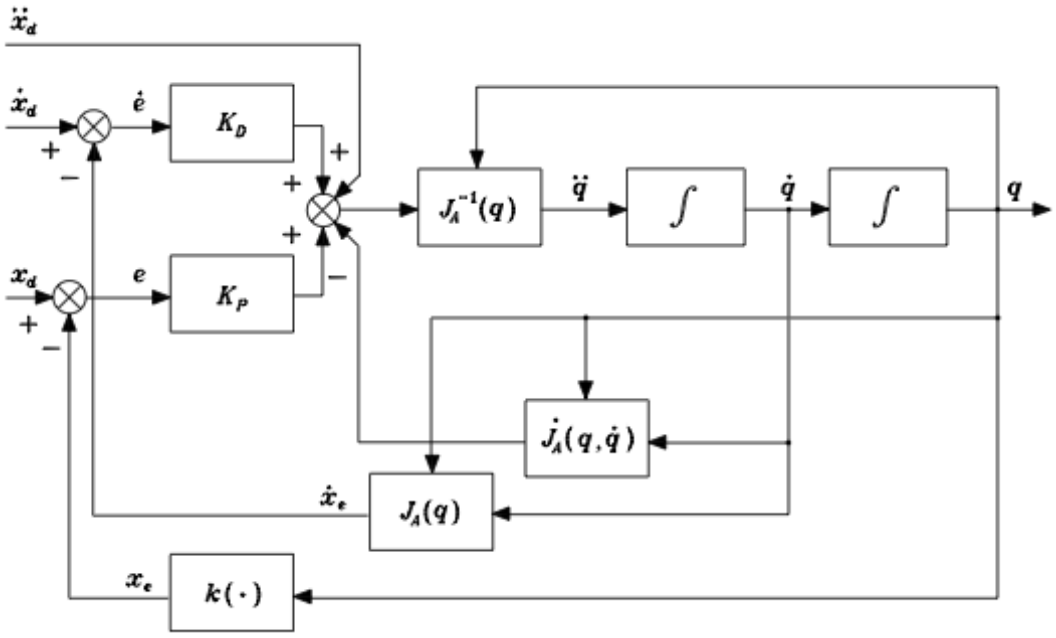



Figura 3.3.2: Algoritmo de inversión cinemática de segundo orden

Para el subprograma de control de manipuladores (Manipulator Control), la función que compila la inversión cinemática es *ik_hoap*.

A continuación se va explicar esta función teniendo presente la ecuación 16 descrita anteriormente.

```
function [q, dq, ddq] = ik_hoap(q0, traj, d_traj, dd_traj, h)
```

Las variables de entrada son la configuración articular en el tiempo inicial (q0), la trayectoria de la posición y orientación (traj), las velocidades (d_traj) , las aceleraciones (dd_traj) y la variable h que contienes las funciones de la librería cinemática creada.

Como salidas, la función *ik_hoap* devuelve el espacio articular (q), el espacio de las velocidades articulares (dq) y el espacio de las aceleraciones articulares (ddq).

En el capítulo siguiente (Diseño de la interfaz) se tratará los tipos de datos y variables empleadas.

En cada iteración del bucle principal:

```
for jj=1:L-1
.....
```

donde L es la longitud del vector tiempo, es decir, el número de puntos que se quiere generar en la trayectoria, se distingue si se está en el caso de doble soporte (case 0), simple soporte en pie derecho (case -1) o simple soporte en pie izquierdo (case 1).

```
switch traj.SF(jj)
   case 0       % double support
      ….
   case -1      % right foot support
      ….
   case 1       % left foot support
end switch
```

Además se tiene en cuenta el movimiento de los brazos.

En cada caso, se quiere hallar la aceleración articular (ddq), y los parámetros que se calculan para lograrlo son:

- El error de la localización deseada frente a la localización actual en cada iteración:

$$e = x_d - x_r$$

En doble soporte se tiene el error del pie derecho (e_RF) y el error del pie izquierdo (e_LF) como:

```
e_RF(:,jj) = evaluate_error (traj.CoM(:,jj)+RF_p0_w,...
       pose_quat2rpy(real(h.RF_T_w(q(:,jj)))));
e_LF(:,jj) = evaluate_error (traj.CoM(:,jj)+LF_p0_w,...
       pose_quat2rpy(real(h.LF_T_w(q(:,jj)))));
```

En simple soporte con el pie derecho los errores de los pies se calculan como:

```
e_RF(:,jj) = evaluate_error (traj.CoM(:,jj)+RF_p0_w,...
       pose_quat2rpy(real(h.RF_T_w(q(:,jj)))));
e_LF(:,jj) = evaluate_error (traj.LF(:,jj)+w_p0_LF,...
             pose_quat2rpy(real(h.w_T_LF(q(:,jj)))))
```

Siendo w_p0_LF (w_p0_RF) la posición del pie izquierdo (derecho) respecto a la cadera justo antes de iniciarse el movimiento de simple soporte.

Del mismo modo, en simple soporte con el pie izquierdo:

```
e_LF(:,jj) = evaluate_error (traj.CoM(:,jj)+LF_p0_w,...
       pose_quat2rpy(real(h.LF_T_w(q(:,jj)))));
```

```
e_RF(:,jj) = evaluate_error (traj.RF(:,jj)+w_p0_RF,...
       pose_quat2rpy(real(h.w_T_RF(q(:,jj)))));
```

En el movimiento de los brazos, estos errores se calculan como:

```
% Right and left arm
    e_RH(:,jj) = evaluate_error (traj.RH(:,jj),...
        pose_quat2rpy(h.CoM_T_RH(q(:,jj))));
    e_LH(:,jj) = evaluate_error (traj.LH(:,jj),...
              pose_quat2rpy(h.CoM_T_LH(q(:,jj))));
```

- Términos u_R, u_L para las piernas

En el caso de doble soporte:

```
u_R = [dd_traj.CoM(1:3,jj);...
       zeros(3,1)] + Kd * [d_traj.CoM(1:3,jj);zeros(3,1)] +
Kp*e_RF(:,jj);
u_L = [dd_traj.CoM(1:3,jj);....
       zeros(3,1)] + Kd * [d_traj.CoM(1:3,jj);zeros(3,1)] +
Kp*e_LF(:,jj);
```

En el caso de simple soporte en pie derecho:

```
u_R = [dd_traj.CoM(1:3,jj);...
       zeros(3,1)] + Kd * [d_traj.CoM(1:3,jj);zeros(3,1)] +
Kp*e_RF(:,jj);
u_L = [dd_traj.LF(1:3,jj);...
       zeros(3,1)] + Kd * [d_traj.LF(1:3,jj);zeros(3,1)] +
Kp*e_LF(:,jj);
```

En el caso de simple soporte en el pie izquierdo:

```
u_L = [dd_traj.CoM(1:3,jj);...
       zeros(3,1)] + Kd * [d_traj.CoM(1:3,jj);zeros(3,1)] +
Kp*e_LF(:,jj);
u_R = [dd_traj.RF(1:3,jj);...
       zeros(3,1)] + Kd * [d_traj.RF(1:3,jj);zeros(3,1)] +
Kp*e_RF(:,jj);
```

- Términos u_RH y u_LH para los brazos:

```
u_RH = dd_traj.RH(:,jj) + Kd * d_traj.RH(:,jj) + Kp * e_RH(:,jj);
u_LH = dd_traj.LH(:,jj) + Kd * d_traj.LH(:,jj) + Kp * e_LH(:,jj);
```

Finalmente la aceleración se halla realizando la siguiente operación con la función interna *invert_kinematic_standard*:

En el caso de doble soporte:

```
ddq(1:6, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.RF_J_w, u_R, 1:6, 1:6);
```

```
ddq(7:12, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.LF_J_w, u_L, 1:6, 1:6);
```

En el caso de simple soporte en pie derecho:

```
ddq(1:6, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.RF_J_w, u_R, 1:6, 1:6);
ddq(7:12, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.w_J_LF, u_L, 1:6, 1:6);
```

En el caso de simple soporte en pie izquierdo:

```
ddq(7:12, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.LF_J_w, u_L, 1:6, 1:6);
ddq(1:6, jj+1) = invert_kinematics_standard (q(:,jj),...
       h.w_J_RF, u_R, 1:6, 1:6);
```

En el movimiento de los brazos, la aceleración de la mano derecha e izquierda se calcula como:

```
ddq(14:18, jj+1) = invert_kinematics_standard (q(:,jj),...
        h.CoM_J_RH, u_RH, 1:3, 1:5);
ddq(19:23, jj+1) = invert_kinematics_standard (q(:,jj),...
        h.CoM_J_LH, u_LH, 1:3, 1:5);
```

Una vez se obtiene la aceleración, se pasa a calcular la velocidad y la localización realizando una integración mediante la función interna *integrate_vector* :

```
dq(:, jj+1)  = integrate_vector (dq(:, jj), ddq(:, jj+1), Ts);
q(:, jj+1)   = integrate_vector (q(:, jj), dq(:, jj+1), Ts);
```

Las funciones internas empleadas son las siguientes:

```
function ddq = invert_kinematics_standard (q_act, J, e, m, n)
J1 = J(q_act);
ddq = J1(m, n)\e(m);
end

function x_next = integrate_vector (x, dx, Ts)
x_next = x + dx * Ts;
end

function error = determine_error (pd, p)
error = pd - p;
end
```

# 3.4 Generación de trayectorias

Una vez planteado y resuelto el problema de la cinemática directa e inversa, se generan las trayectorias en el espacio operacional. El concepto de trayectoria que el presente proyecto expone, consiste en el conjunto de leyes de tiempo de aquellas variables de posición y orientación que describen la localización del extremo del manipulador a controlar (brazos ó piernas).

En todas las trayectorias propuestas necesitamos unos inputs que serán la base de tiempo, tiempos inicial y final, incrementos de las posiciones inicial y final, así como velocidades y aceleraciones iniciales y finales dependiendo de tipo de trayectoria. Consideramos que la ley del tiempo es lineal:

$$\Delta t = t_f - t_0$$

$$t_{i+1} = t_i + \Delta t$$

## 3.4.1 Trayectoria lineal

Dadas una localización inicial (posición y orientación) y final, se puede generar la trayectoria mediante la relación lineal de tiempo siguiente:

$$x_i = x_0 + \frac{x_f - x_0}{t_f - t_0} \cdot t_i$$

**Ecuación 18**

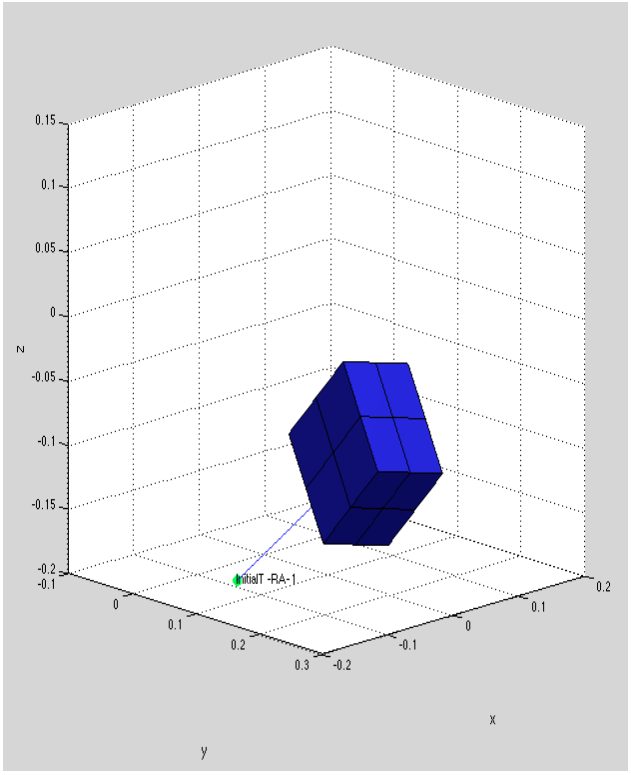



Figura 3.4.1.1: Trayectoria lineal

En esta figura se puede ver representado el extremo del manipulador mediante el cubo de color azul que recorre la trayectoria lineal desde el punto inicial (en verde) hasta el punto final (en rojo, sin apreciar en esta figura).

O bien mediante incrementos de posiciones y orientaciones:

$$\Delta x = x_f - x_0 \qquad \Delta\alpha = \alpha_f - \alpha_0$$

$$x_i = x_0 + \frac{\Delta x}{\Delta t} \cdot t_i \qquad \alpha_i = \alpha_0 + \frac{\Delta\alpha}{\Delta t} \cdot t_i$$

De forma matricial para implementar la trayectoria con Matlab se puede definir el problema de la siguiente forma:

```
CM=zeros(2,n);
M=[t1 1;t2 1];
DM=[p0;p1];
for i=1:n
    CM(:,i)=M\DM(:,i);
    x(i,:)=([tt' ones(L,1)]*CM(:,i))';

end
```

Donde: $DM = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$ siendo

$$p_0 = (x_0, y_0, z_0, roll_0, pitch_0, yaw_0)$$

$$p_1 = (x_1, y_1, z_1, roll_1, pitch_1, yaw_1)$$

Para el caso general de espacio de trabajo (operacional) de seis dimensiones, n = 6, realizando el bucle para las tres coordenadas de posición y las tres de orientación.

Se debe tener en cuenta que esta trayectoria presenta discontinuidad en la velocidad entre dos trayectorias que se deseen juntar, por lo tanto no se podrá controlar la velocidad.

## 3.4.2 Trayectoria circular

En primer lugar, se ha incluido en la generación de trayectorias, el arco entre dos puntos teniendo un punto intermedio. Se ha decidido darle el nombre de trayectoria circular para facilitar al usuario de la interfaz una opción más en las interpolaciones. Pero esta interpolación se trata de una ruta o camino donde no se dispone de información de tiempos, y por tanto no es una trayectoria como tal por definición.

Para esta trayectoria se necesita definir un punto inicial (P1) y final (P3) y un punto intermedio vía (P2) que se haya en el camino deseado. Se trata de buscar el centro y el radio del "posible" arco que pasa por los tres puntos. Remarcar <<posible>>, porque no siempre se podrá hallar la solución a este problema. El centro se halla mediante la intersección de las mediatrices de los segmentos P1P2 y P2P3, el llamado circuncentro de un triángulo. Con el siguiente gráfico se puede entender mejor el problema.

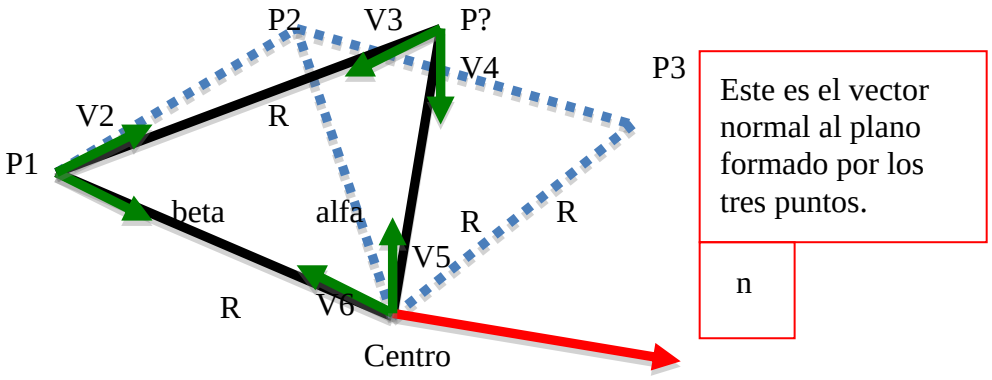



Figura 3.4.2.1 Vector normal al plano formado por 3 puntos

Hallar el vector normal al plano formado por los tres puntos vendrá explícitamente del determinante siguiente:

Encontrar la solución a esta trayectoria no es fácil y para ello se requiere resolver tres sistemas implementando el siguiente algoritmo en Matlab:

```
function [C,r,plane,n]=circum(p1,p2,p3)
%
% FUNCTION TO WORK OUT Center, Radius and Plane
%

% Matrix of points
MP = [p1;p2;p3];
%Matrix of medium points
MPM = [p1 + (p2-p1)/2;p3 + (p2-p3)/2];

% Plane of 3 points
syms x y z real
vsym = [x y z];
plane = det([vsym-p1;p2-p1;p3-p1]);

% normal vector of the plane
 n=[diff(plane,x) diff(plane,y) diff(plane,z)];

% directional vectors
v=[p2-p1;p2-p3];

% Matrix of p2 vs p1
MVi0=[n;v(1,:)];

% Matriz of p2 vs p3

MVi1=[n;v(2,:)];

% Normal vector to n and v1
syms x1 y1 z1 real
vsym1=[x1 y1 z1];
Sistema1=MVi0*vsym1';
% Solution system1
x1=1;
[sol1,sol2]=solve(eval(Sistema1));
vp3=[1 eval(sol1) eval(sol2)];

% Normal vector to n and v2
syms x2 y2 z2 real
vsym2=[x2 y2 z2];
Sistema2=MVi1*vsym2';
% Solution system2
x2=1;
[sol3,sol4]=solve(eval(Sistema2));
vp2=[1 eval(sol3) eval(sol4)];

MVP=[vp3;vp2];
% Solution of centre
syms r s real
Sistema3=MPM(1,:)+r*MVP(1,:)-MPM(2,:)-s*MVP(2,:);
[r,s]=solve(Sistema3(1),Sistema3(2));
```

```
C=eval(sym(MPM(1,:)+r*MVP(1,:)));

% Matrix with distance point vs centre
MR=[norm(MP(1,:)-C);norm(MP(2,:)-C);norm(MP(3,:)-C)];
r=MR(1);
```

La trayectoria circular la utilizará la interfaz para el movimiento de los brazos. En la figura siguiente se puede ver cómo el cubo, que representa al extremo del manipulador, recorre un arco entre los puntos inicial (en verde) y final (en rojo) pasando por un punto vía ( en este caso el punto es el [0,0,0]).

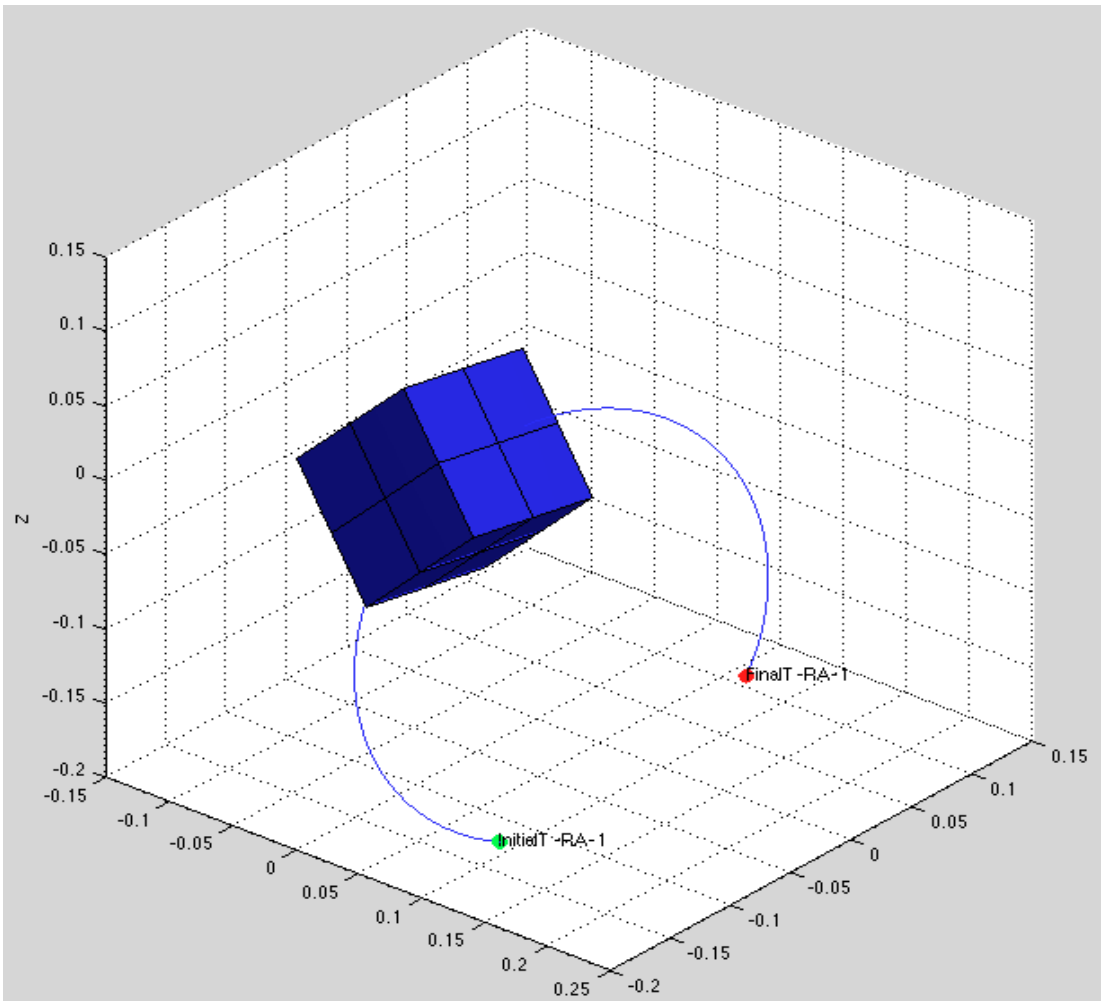



Figura 3.4.2.1: Trayectoria circular

## 3.4.3 Trayectoria polinómica

La utilización de polinomios en la generación de trayectorias se debe a la linealidad de los sistemas que forman para hallar los parámetros de estos polinomios en función de la variable de tiempo.

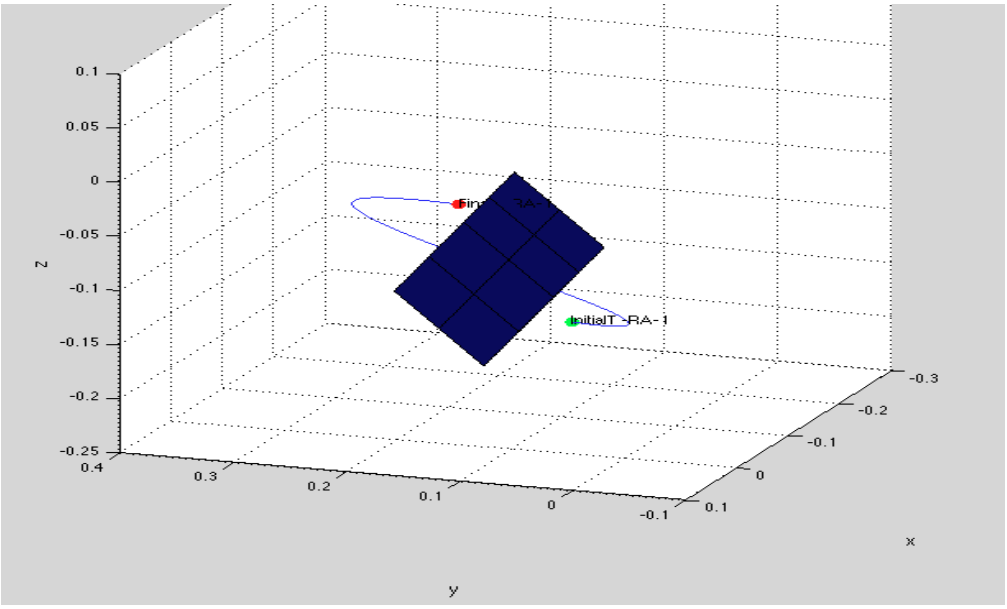



Figura 3.4.3: Trayectoria polinómica

En la figura 3.4.3 se puede observar nuevamente un paralelepípedo de color azul que recorrer una trayectoria polinómica desde el punto inicial (verde) al punto final (rojo), siendo esta trayectoria de tercer orden. Por ello los inputs que se requieren tener son la localización y velocidad inicial y final. Otra ventaja de esta trayectoria es que se tiene un polinomio para cada coordenada.

Sea: $$u_t = a_0 + a_1 \cdot t + a_2 \cdot t^2 + a_3 \cdot t^3$$

Considérese las condiciones iniciales y finales para la posición y velocidad

$$u(0) = u_0$$
$$u(t_f) = u_f$$

$$\dot{u}(t) = a_0 + 2 \cdot a_2 \cdot t + 3 \cdot a_3 \cdot t^2$$

$$\dot{u}(0) = 0 \qquad \dot{u}(t_f) = 0$$

$$\ddot{u}(t) = 2 \cdot a_2 + 6 \cdot a_3 \cdot t$$

$$\ddot{u}(t) = 6 \cdot a_3$$

$$u(t) = u_0 + \frac{3}{t_f^2} \cdot (u_f - u_0) \cdot t^2 + \left(\frac{2}{t_f^3}\right) \cdot (u_f - u_0) \cdot t^3$$

La ventaja de la trayectoria polinómica de tercer orden frente a las anteriores es la continuidad que muestra en la posición y en la velocidad. Aún así, sigue sin tener continuidad en la aceleración y esto es importante ya que no es bueno que existan picos de aceleración por razones mecánicas.

Parte de la función implementada para esta trayectoria muestra el bucle matricial para hallar los parámetros de los polinomios:

```
CM=zeros(4,6);
M=[t0^3 t0^2 t0 1;T^3 T^2 T 1;3*t0^2 2*t0 1 0;3*T^2 2*T 1 0];
MI=inv(M);
DM=[p0;p1;v0;v1];
for i=1:6
    CM(:,i)=MI*DM(:,i);
    Points(i,:)=[[[t.^3]', [t.^2]', t', ones(n,1)]*CM(:,i)]';
    VPoints(i,:)=[[[3*t.^2]' [2*t]' ones(n,1) zeros(n,1)]*CM(:,i)]';
    APoints(i,:)=[[[6*t]' 2*ones(n,1) zeros(n,1)
zeros(n,1)]*CM(:,i)]';
end
x_traj    = create_trajectory_structure(Points, Ts,[t0 T]);
dx_traj   = create_trajectory_structure(VPoints,  Ts,[t0 T]);
ddx_traj  = create_trajectory_structure(APoints, Ts,[t0 T]);
```

### 3.4.4 Trayectoria spline

La trayectoria spline se basa en un polinomio de grado cinco. Además de las posiciones y velocidades, requiere de las aceleraciones inicial y final. Adicionalmente se consideran dos puntos adicionales en la variable tiempo, uno a un tercio del tiempo total y el segundo a dos tercios. Con esto se obliga a que la trayectoria tenga continuidad en la aceleración y la solución del sistema sea más factible.

Se ha empleado la herramienta de Matlab Symbolic Math para conseguir los parámetros de los polinomios.

Para más detalle, véase el anexo con el código fuente de la función *spline_interpolation.m*

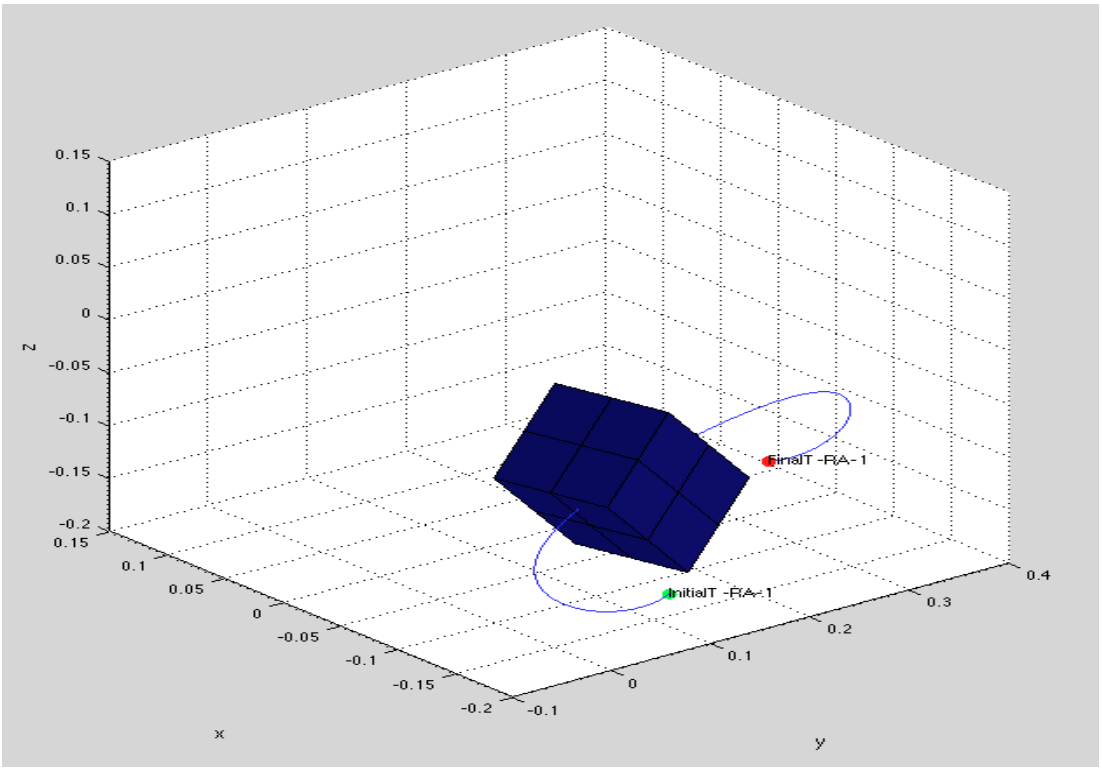



Figura 3.4.4.1: Trayectoria Spline

La figura 3.4.4.1 muestra una trayectoria spline ejemplo, recorrida por el cubo de color azul desde el punto de color verde al punto de color rojo.

# Capítulo 4

# Diseño de la interfaz

# 4.1 Introducción a MATLAB

La interfaz que se presenta en este proyecto ha sido desarrollada en el lenguaje de programación MATLAB.

MATLAB es un lenguaje de alto nivel para realizar cálculos cientifico-técnicos. Integra las herramientas de cálculo necesarias con otras de visualización así como, un entorno de programación de fácil uso.

Propiedad de Mathworks, no es un software gratuito pero desde hace ya unos años está presente en todas las aulas universitarias del ámbito de la ingeniería, ciencias y economía, y destaca por su facilidad y eficiencia en el manejo de vectores y matrices.

Las aplicaciones típicas a destacar son:

- Cálculo matemático
- Desarrollo de algoritmos
- Adquisición de datos
- Modelado, simulación y protipo
- Análisis de datos y visualización
- Gráficos
- Desarrollo de aplicaciones e interfaces gráficas de usuario (GUI)

A su vez, MATLAB presenta una amplia variedad de *toolboxes* y aplicaciones específicas para distintas áreas tecnológicas y científicas.
Estos paquetes incluyen librerías de funciones MatLab (M-files) que extienden las posibilidades de MatLab para resolver problemas específicos

Acompañando a MATLAB, hay que destacar la aplicación SIMULINK que es una herramienta para modelado, simulación y análisis de sistemas dinámicos. Soporta tanto sistemas lineales como no lineales: en tiempo continuo, muestreados, híbridos y sistemas multifrecuencia.

Durante la fase de desarrollo de la interfaz se ha empleado la versión de Matlab 7.12 (R2011a) compatible para el sistema Mac OS X y para Windows. También se ha comprobado los resultados de simulación de ciertas trayectorias en un modelo dinámico realizado por Paolo Piero en Simulink.

# 4.2 Diseño de la interfaz

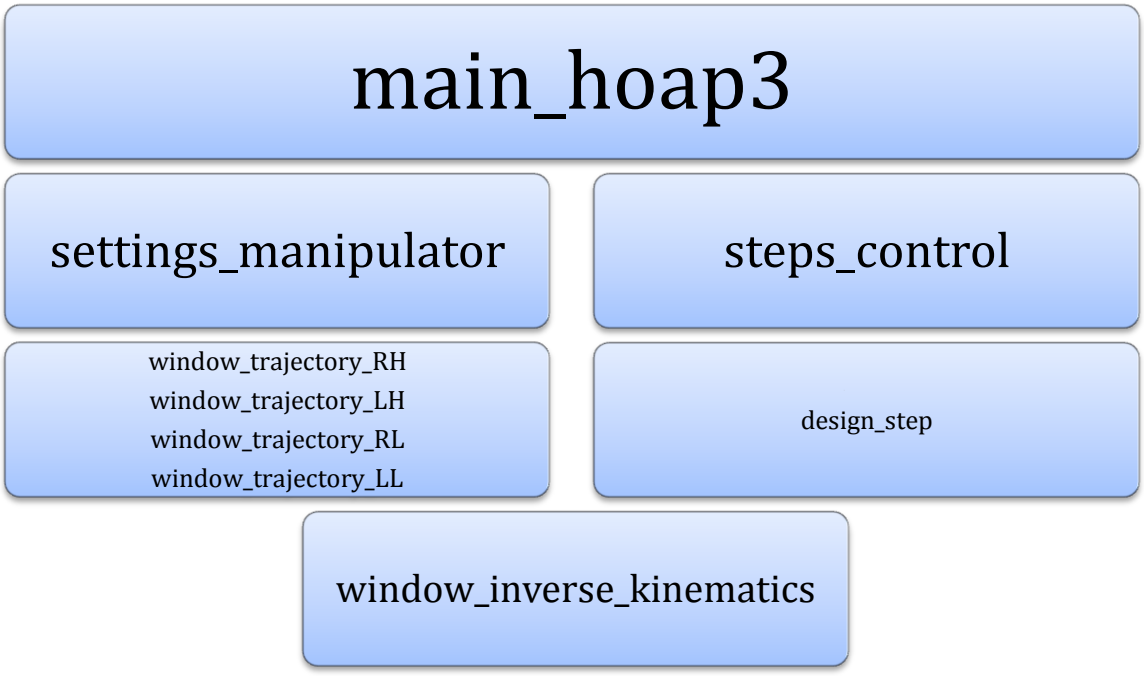
Para el diseño e implementación de la interfaz se ha empleado la herramienta GUIDE de Matlab que permite un acceso total e intuitivo a los componentes de la interfaz facilitando el diseño desde el inicio. Al crear una ventana se genera automáticamente los archivos **.m* y **.fig* de la misma. Esto supone una potente herramienta para cualquier tipo de usuario de Matlab .

Se ha decidido por una programación lineal y puntualmente dirigida a objetos, para obtener una interfaz simple e intuitiva. Hay ciertos elementos, como las pestañas que se crean programáticamente desde las funciones de las ventanas. Estas pestañas(tabs) son resultado de código abierto compartido en internet por Shiying Zhao (zhao@arch.umsl.edu) en el año 2006, no son funciones oficiales de MATLAB ni comentadas en el manual de ayuda.

En los próximos apartados se va explicar la estructura del programa, las funciones principales así como las funciones dedicadas a la representación gráfica y creación de objetos para su fin.

## 4.2.1 Estructura del programa

Dado que la interfaz se ha direccionado hacia un control de bajo y alto nivel, la estructura se divide en dos ramas como se puede ver en la figura de abajo.



La estructura de datos empleada en cada subprograma es la siguiente.

Para el subprograma **Manipulator Control** se ha empleado cinco variables globales:

global SETTINGS_hoap manipulator_data Trajectory d_Trajectory dd_Trajectory

**SETTINGS_hoap** : es una variable de tipo estructura (struct) con los siguientes campos:

*units*: variable de tipo estructura destinada a almacenar las unidades de la posición (metros ó milímetros) y orientación (radianes ó grados), así como el tipo de localización empleada(diferencial ó absoluta). Los campos de esta variable son:

*pos*: almacena un string (“m” ó “mm”)

*orient*: almacena un string (“rad” ó “def”)

*def*: almacena un string(“differential” ó “absolute”)

*parameters*: variable de tipo estructura que almacena los parámetros del tiempo de paso (Ts en segundos) y de inversión cinemática (Kp y Kd)

*save*: variable de tipo estructura que almacena si se quiere salvar los resultados en formato *.txt o *.csv.

*initial*: variable de tipo estructura con los siguientes campos.

*from*: almacena en un string si la configuración inicial de los ángulos del robot viene de un archivo o se utiliza la configuración por defecto.

*data*: almacena en un array vector los 23 valores de los ángulos de cada articulación.

*manipulators2use*: variable de tipo estructura que almacena qué manipuladores se van a emplear en las ventanas futuras.

*h:* variable tipo estructura que contiene la librería de funciones de la cinemática del robot humanoide HOAP-3.

*hoap3:* variable las estructura de datos del robot HOAP-3 con todas las características geométricas y cinemáticas.

*humanoid_fields:* variable de tipo estructura con los principales campos a emplear en los manipuladores.

**manipulator_data**: variable de tipo estructura con los campos correspondientes a los manipuladores que se pueden presentar en el robot:
RA : Right Arm
RLS: Right Leg Support
RLF: Right Leg Floating
LA: Left Arm
LLS: Left Leg Support
LLF: Left Leg Floating

Cada campo de esta estructura a su vez es una variable de tipo estructura con los siguientes campos:

*datas*: variable de tipo "cell" con cuatro filas y dos columnas. Dos columnas porque cómo máximo habrá dos trayectorias por manipulador. Cuatro filas porque en cada una se almacenará:

Fila 1: matriz double con seis filas y seis columnas. Seis filas correspondientes a las variables x,y,x,roll,pitch y yaw. Seis columnas correspondientes a la localización inicial(columna 1) y final (columna 2), velocidad inicial y final, aceleración inicial y final respectivamente.

Fila 2: variable array double con una fila y dos columnas que almacena el tiempo inicial y final respectivamente.

Fila 3: variable tipo lista y string que almacena los tipos de interpolación que tienen la posición y orientación ("Linear","Polinomic","Circular","Spline")

Fila 4: variable tipo string que almacena el nombre de la trayectoria.

*count_traj*: variable array numérica de dimensiones 1x2 que contendrá unos o ceros. Si existe una primera trayectoria para ese manipulator tendrá un 1 en su primera columna, ó 0 en caso de no haber trayectoria primera.

*W_xx:* donde xx será las iniciales anteriores correspondiente a cada tipo de manipulador que se tendrá (RA, LA, RLS, etc). En esta variable se almacenará la estructura que contiene la ventana del manipulador que representa.

**Trajectory , d_Trajectory, dd_Trajectory**

Cada una de estas variables tipo estructura es la trayectoria del espacio de trabajo, del espacio de velocidades y del espacio de aceleraciones respectivamente. Poseen los siguientes campos:

*CoM*: Center of Mass. Variable tipo array 6xn que contiene la localización del centro de masas respecto a su posición inicial (0,0,0) en cada instante de tiempo.

*RF*: Right Foot. Variable array 6xn que contiene la localización del pie derecho respecto a ese centro virtual (0,0,0) en cada instante de tiempo cuando se trate de

pie soporte. Cuando sea pie flotante contendrá el incremento de posición y orientación respecto de la localización en el momento en que se produzca el cambio de soporte a flotante.

*LF*: Left Foot. Misma variable que RF pero referida al pie izquierdo.

*RH*: Right Hand. Variable array 6xn que contiene la localización de la mano derecha respecto al centro de masas en cada instante de tiempo.
*LH*: Left Hand. Variable array 6xn que contiene la localización de la mano derecha respecto al centro de masas en cada instante de tiempo.

*SF*: Supporting Foot. Variable array 1xn que contiene 0, 1 o -1 en cada instante de tiempo. Esta variable permite discernir si el robot se encuentra en doble soporte (0), en simple soporte en pie derecho (-1) ó en simple soporte en pie izquierdo (1).

*Ts*: tiempo muestreo en segundos. Será una constante para todas las trayectorias.
time: vector tiempo. Contiene la variable tiempo.

*T*: duración de la trayectoria en segundos. Será una variable de dimensiones 1x1.

Las variables de tipo estructura destinadas a almacenar la trayectoria se emplearán también en el subprograma Step Control.

Para el subprograma **Step Control** se ha empleado tres variables globales y a diferencia del subprograma anterior, al utilizar menos ventanas, se ha utilizado la forma de almacenamiento de datos en la variable *handles* mediante la función de MATLAB *guidata()*.

global h hoap3 pvia

***h*** : esta variable de tipo struct contiene la librería de funciones de la cinemática del robot HOAP-3. Esta librería se carga al inicio del subprograma y es la misma que se emplea en Manipulator Control.

***hoap3*** : variable de tipo struct que contiene las principales características geométricas, cinemáticas y dinámicas del robot Hoap3.

***pvia*** : variable array de tipo doble y dimensiones 6x1 que contiene la localización de un punto vía para la trayectoria circular empleada en la trayectoria de los brazos.

### 4.2.2 Principales funciones creadas en Manipulator Control

En todas las ventanas generadas existen dos funciones de apertura que definen las variables y objetos principales que el programa necesita inicialmente.
La primera de estas dos funciones es:

```
function varargout = settings_manipulator(varargin)
```

Esta primera función es idéntica en todas las ventanas. Define la variable gui_State como variable tipo struct en la cual se almacena información de la función como nombre, función de apertura y cierre y llamadas a dicha función.

La segunda función para la ventana settings_manipulator:

```
% --- Executes just before settings_manipulator is made visible.
%                                                 ---------
settings_manipulator
function settings_manipulator_OpeningFcn(hObject, eventdata,
handles, varargin)
% -----------------
% Initial function
% -----------------
global SETTINGS_hoap
 ..............
guidata(hObject, handles);
```

Para las ventanas window_trajectory la función inicial de apertura será la siguiente:

```
% --- Executes just before window_trajectory_RA is made visible.
function window_trajectory_RA_OpeningFcn(hObject, eventdata,
handles, varargin)

global SETTINGS_hoap manipulator_data Trajectory d_Trajectory
dd_Trajectory
 ............
guidata(hObject, handles);
```

En estas funciones de apertura, se definen todos los objetos que van aparecer en la ventana, así como las variables y estructuras se inicializan. Aquellas que se almacenan en la estructura por defecto de la ventana, *handles,* para que los cambios realizados en cada función tengan efecto debe añadirse esta línea de código al final de cada función:

```
guidata(hObject, handles);
```

Dicha línea llama a la función interna de Matlab guidata, y le dije que en el objeto actual actualice todos los campos de la variable handles.

Destacar la función que permite crear las pestañas en los paneles *uitabpanel*: esta función crea un objeto que contiene las pestañas o subpaneles en los cuales se pueden agrupar bajo la herencia de cada pestaña otros objetos de la interfaz.

La función que realiza la inversión cinemática en Manipulator Control es ***ik_hoap***

```
function [q, dq, ddq] = ik_hoap(q0, traj, d_traj, dd_traj, h)
%
%
% Inverse Kinematics Algorith
%
% Author: P.Pierro
% Version: Daniel J. GarcÌa-Cano Locatelli
```

Esta función recibe como parámetros de entrada:

```
ik_hoap(q0, traj, d_traj, dd_traj, h)
```

q0 : ángulos iniciales de los motores de todas las articulaciones.

traj : estructura de datos que contiene la trayectoria en el campo de trabajo. Inicialmente tendrá la localización inicial de los extremos de cada manipulador y del centro de masas.

d_traj : estructura de datos que contiene las velocidades de la trayectoria.

dd_traj : estructura de datos que contiene las aceleraciones de la trayectoria.

h : variable estructura con la librería de funciones de la cinemática del robot hoap-3.

El output de esta función

```
[q, dq, ddq] = ik_hoap
```

q : matriz de 23 filas y n columnas. Siendo n el número de iteraciones o instantes de tiempo de la trayectoria. Contiene en cada columna los 23 ángulos de las articulaciones del robot. El cambio de la notación y orden de estos 23 ángulos respecto de la tabla 2.2.1.1 se hizo para facilitar la programación y comprensión del algoritmo de inversión cinemática. Véase la siguiente tabla.

| Índice qi | Articulación | Movimiento |
|---|---|---|
| q1 | RLEG_JOINT 1 | Right hip joint torsion |
| q2 | RLEG_JOINT 2 | Right hip joint roll |
| q3 | RLEG_JOINT 3 | Right hip joint pitch |
| q4 | RLEG_JOINT 4 | Right knee |
| q5 | RLEG_JOINT 5 | Right ankle pitch |
| q6 | RLEG_JOINT 6 | Right ankle roll |
| q7 | LLEG_JOINT 1 | Left hip joint torsion |
| q8 | LLEG_JOINT 2 | Left hip joint roll |
| q9 | LLEG_JOINT 3 | Left hip joint pith |
| q10 | LLEG_JOINT 4 | Left knee |
| q11 | LLEG_JOINT 5 | Left ankle pitch |
| q12 | LLEG_JOINT 6 | Left ankle roll |
| q13 | BODY_JOINT 1 | Waist pitch |
| q14 | RARM_JOINT 1 | Right shoulder pitch |
| q15 | RARM_JOINT 2 | Right shoulder roll |
| q16 | RARM_JOINT 3 | Right shoulder torsion |
| q17 | RARM_JOINT 4 | Right elbow |
| q18 | RARM_JOINT 5 | Right hand torsion |
| q19 | LARM_JOINT 1 | Left shoulder pitch |
| q20 | LARM_JOINT 2 | Left shoulder roll |
| q21 | LARM_JOINT 3 | Left shoulder torsion |
| q22 | LARM_JOINT 4 | Left elbow |
| q23 | LARM_JOINT 5 | Left hand torsion |

dq: matriz de velocidades angulares de dimensión 23xn.

ddq : matriz de aceleraciones angulares de dimensión 23xn.

### 4.2.3 Principales funciones creadas en Step Control

La segunda parte de la interfaz de generación de trayectorias tiene la ventana de insercción de parámetros del paso a generar y la venta de diseño del paso.

En la ventana de diseño de paso también se ha colocado un panel realizado con la función uitabpanel, pero a diferencia de las ventanas de Manipulator Control, este panel es del tipo popup y contiene las gráficas de los ángulos, velocidades angulares y/ó aceleraciones angulares de las articulaciones de los manipuladores.

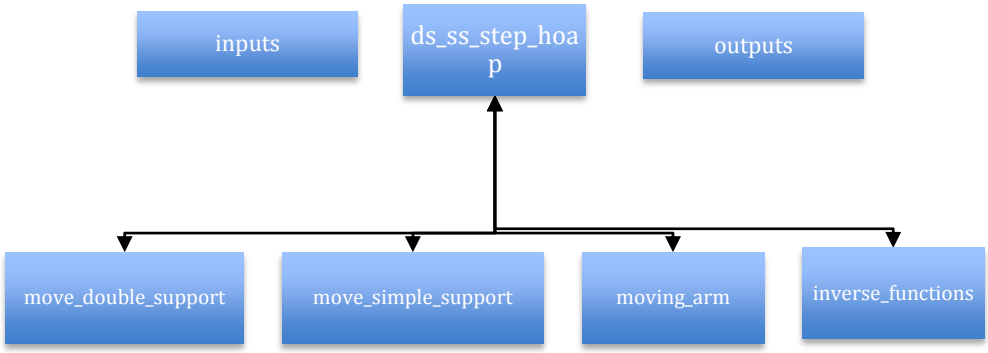
Las principales funciones a destacar son:

***ds_ss_step_hoap*** : esta función se llama al pulsar el botón GENERATE de la ventana Step Design. Es la función más importante pues coordina todo el proceso de generación de trayectoria.

```
function [q,dq,ddq,trajectory,d_trajectory,dd_trajectory] =
ds_ss_step_hoap(delta,data,leg)
```

Como datos de entrada recibe los incrementos de posición de cada manipulador así como los tipos de interpolación que hay que realizar en la variable *delta*. En la variable de entrada *data* , de tipo estructura, vendrán almacenados aquellos principales datos del paso como son la longitud y altura del mismo, tiempo de paso, duración de la trayectoria, configuración de los ángulos iniciales de las articulaciones, si se está en doble o simple soporte, etc. La última variable de entrada que recibe esta función, *leg*, indicará qué pierna hace de soporte.

La salida de esta función son las matrices q, dq, ddq que contienen los ángulos, velocidades y aceleraciones angulares de los motores de los manipuladores del robot, y las variables de tipo estructura trajectory, d_trajectory, dd_trajectory que contienen la trayectoria de la localización, velocidad y aceleración del centro de masas, pie derecho e izquierdo y brazo derecho e izquierdo, así como la variable instante de tiempo time.




***move_double_support*** : esta función se encarga de interpolar la trayectoria del centro de masas en el movimiento de doble soporte. En este movimiento los dos pies está fijos y el punto que se estudia su trayectoria es el CoM (Center of Mass). De las variables de entrada de esta función cabe destacar:

delta_p : variable que contiene el incremento de la posición del centro de masas. El incremento en la dirección x e y se calculan de esta manera:

```
switch Leg
    case 'Right Leg Support' % Support on right foot
        delta = h.CoM_T_RF(q0);
        delta_P(1,:) = (2*delta(1) +
(hoap3.legs.right.foot.limits.x(1) +
hoap3.legs.right.foot.limits.x(2))/1);
        delta_P(2,:) = (2*delta(2) +
(hoap3.legs.right.foot.limits.y(1) +
hoap3.legs.right.foot.limits.y(2))/1);

    case 'Left Leg Support' % Support on left foot
        delta = h.CoM_T_LF(q0);
        delta_P(1,:) = (2*delta(1) +
(hoap3.legs.left.foot.limits.x(1) +
hoap3.legs.left.foot.limits.x(2))/1);
        delta_P(2,:) = (2*delta(2) +
(hoap3.legs.left.foot.limits.y(1) +
hoap3.legs.left.foot.limits.y(2))/1);

end
```

Y se dejan fijos sin que el usuario pueda modificarlos, debido al criterio de estabilidad en el que está basado este cálculo. Mientras que el incremento en la dirección z se deja libre para que el usuario decida cuanto baja o sube el CoM.

```
function [trajectory, d_trajectory, dd_trajectory] =
move_double_support (delta_p, Ts, T, trajectory, d_trajectory,
```

```
dd_trajectory,interpola)
```

***move_simple_support*** : función que se encarga de la interpolación del centro de masas y del pie flotante en el movimiento de simple soporte. En este movimiento la trayectoria tiene dos fases. La primera desde el tiempo inicial al tiempo en el cual se alcanza la máxima altura del paso. La segunda correspondiente a este último tiempo hasta el tiempo final del paso.
Por ello, se tendrá como variables a destacar de entrada, los incrementos de la posición del centro de masas *delta_com* y los incrementos de la posición del pie flotante *delta_FF* ambos en las dos fases.

```
function [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_com, delta_FF, Ts, T, trajectory,
d_trajectory, dd_trajectory, leg,interpolaCoM,interpolaFF)
```

***moving_arm*** : esta función se encarga de la interpolación de las trayectorias que seguirán los extremos de cada brazo.

```
function [traj, d_traj, dd_traj] = moving_arm
(manipulator,delta_A,d_delta_A,dd_delta_A,Ts, T, traj, d_traj,
dd_traj,interpola)
```

***inverse_functions*** : se ha creado una función de inversión para cada caso que se pueda presentar en la generación de un paso.

Pie derecho soporte con un movimiento de doble y simple soporte:

```
function [q, dq, ddq] = inverse_right_ds_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h)
```

Pie derecho soporte con un movimiento de simple soporte:

```
function [q, dq, ddq] = inverse_right_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h)
```

Pie izquierdo soporte con un movimiento de doble y simple soporte:

```
function [q, dq, ddq] = inverse_left_ds_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h)
```

Pie izquierdo soporte con un movimiento de simple soporte:

```
function [q, dq, ddq] = inverse_left_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h)
```

### 4.2.4 Funciones de representación gráfica

Para terminar este capítulo de diseño de la interfaz, se va a explicar brevemente las principales funciones creadas para representar gráficamente los resultados.

Función *plot_movies* : con esta función se genera un objeto gráfico que representa al extremo del manipulador que se esté estudiando en ese momento. Este objeto se moverá a lo largo de la trayectoria generada representada en una grafica 3D y también irá cambiando su orientación a lo largo de la misma. El tiempo de representación no corresponde a un tiempo real debido a que la velocidad de procesamiento de todos los puntos del objetos no es la idónea.

```
function plot_movies(data,colord,colname)
```

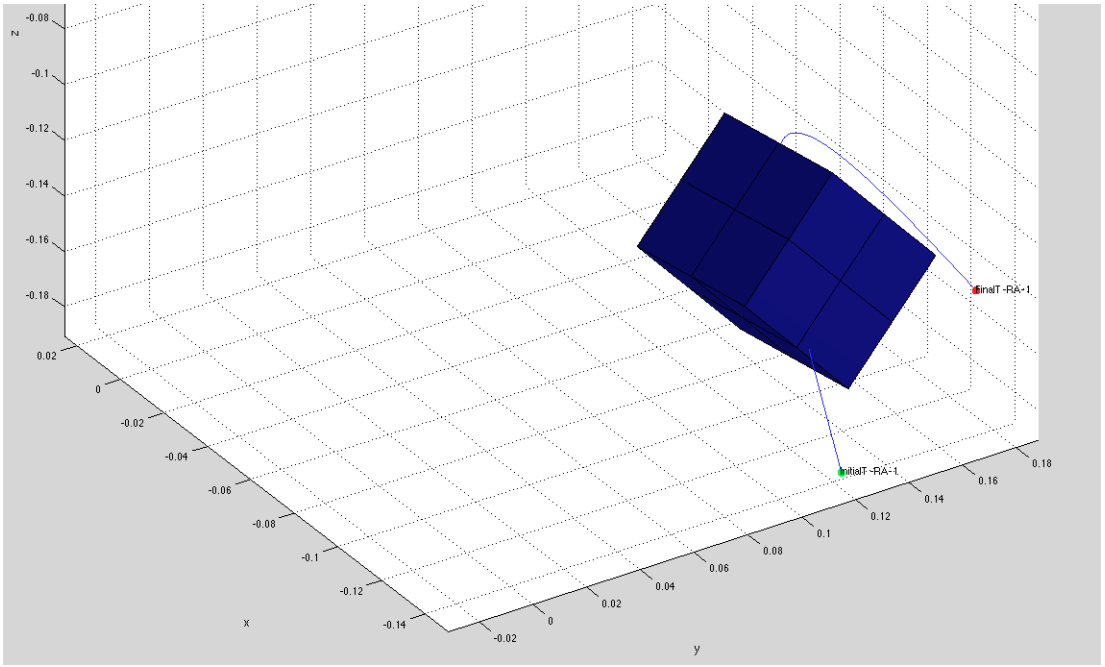



Figura 4.2.4.1: Trayectoria representada con plot_movies

Función *prueba_ZMP* : con esta otra función se ha pretendido representar el movimiento de los pies durante el paso generado así como el movimiento del dentro de masas.

```
function prueba_ZMP(handles)
```

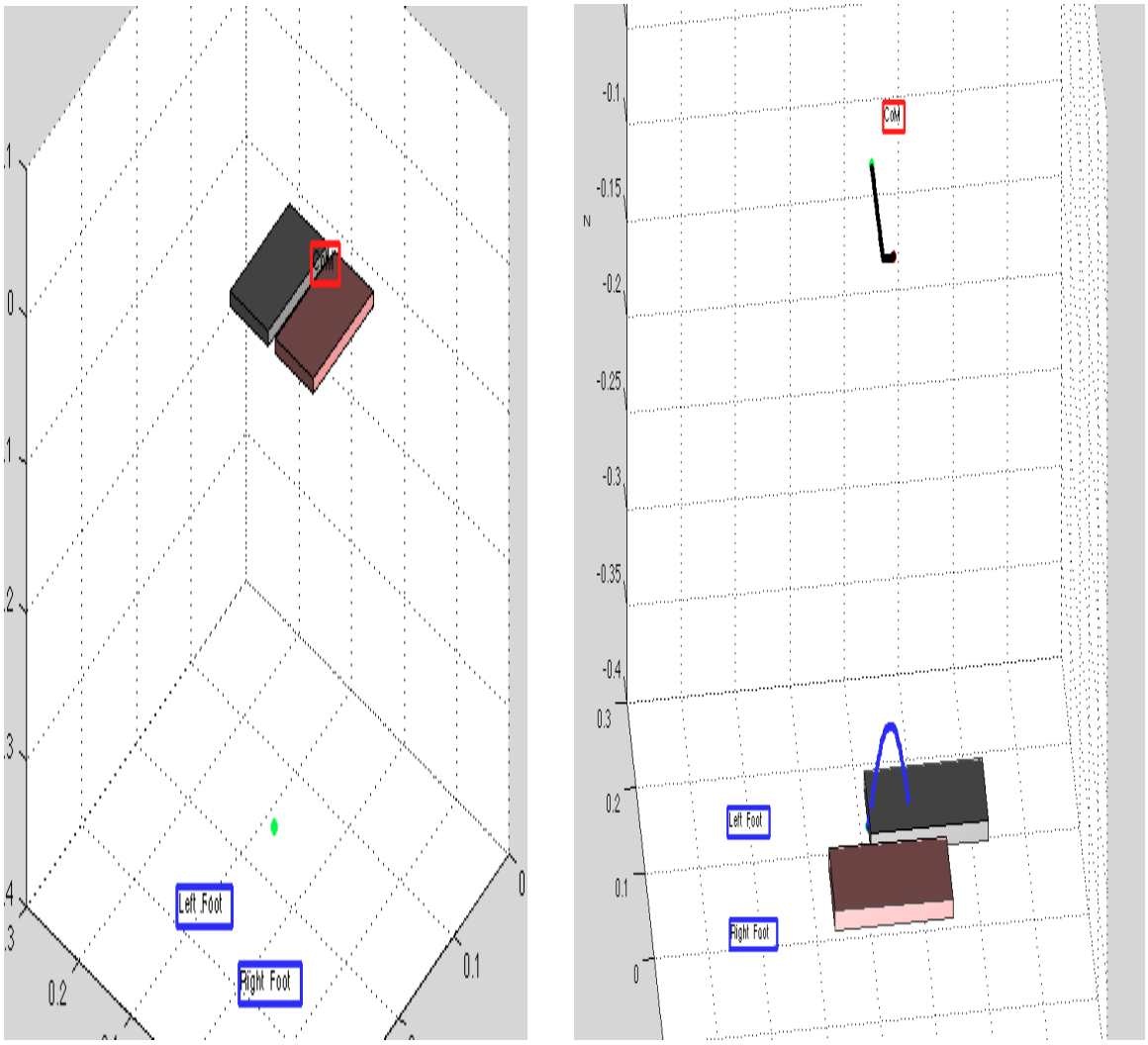



Figura 4.2.4.2 :Ejemplos de representación de un paso con la función prueba_ZMP

# Capítulo 5

# Funcionamiento de la interfaz

# 5.1 Introducción

La interfaz ha sido diseñada pensando en un usuario con conocimientos técnicos pero que no tiene porqué tener conocimientos de robótica o programación para manejar el software. Desde un principio se pensó en hacerla lo más funcional posible y con dos claras aplicaciones:

**MANIPULATORS CONTROL**: dedicada a un estudio y control de bajo nivel de los manipuladores para poder generar trayectorias en el espacio operacional y calcular las configuraciones angulares del robot en cada instante de tiempo.

**STEPS CONTROL**: enfocada a un control de alto nivel para poder generar trayectorias globales, esto es, generar un paso.


Figura 5.1.1: Aspecto de la pantalla inicial de la interfaz

# 5.2 Ejecutable

La interfaz se podrá ejecutar en equipos dispongan de MATLAB o no. Es compatible con sistemas operativos Windows de 64 y 32 bits y con Mac OS, aunque hay que destacar que en estos últimos no hay ejecutables como tales y solo se podrá tener acceso a la interfaz a través del programa MATLAB para Mac OS.

Para equipos Windows con Matlab instalado será suficiente con ejecutar el archivo “Main_Hoap3.exe” que se encuentra en la carpeta del programa adjunto.

Para equipos Windows que no dispongan de Matlab hay que instalar el MCR (MATLAB Compiler Runtime) que viene adjunto para que la interfaz puede hacer uso de las librerías del programa.

# 5.3 Manipulator Control

## 5.3.1 Introducción al control del manipulador

El usuario podrá diseñar trayectorias para cada manipulador de manera aislada, como si se tratase de un <<laboratorio de trayectorias>>. Es un control de bajo nivel que permite estudiar individualmente la trayectoria en el espacio de trabajo del extremo de cada manipulador, realizar la inversión cinemática del manipulador aislado o realizarla de forma completa en el robot entero.

Se podrán generar dos trayectorias consecutivas, estudiar la evolución en el tiempo de cada coordenada de posición y orientación, y simular el movimiento del extremo del manipulador en el espacio.

## 5.3.2 Configuración de parámetros

La primera ventana con la que se encontrará el usuario será la de configuración de los parámetros del subprograma Manipulator Control.

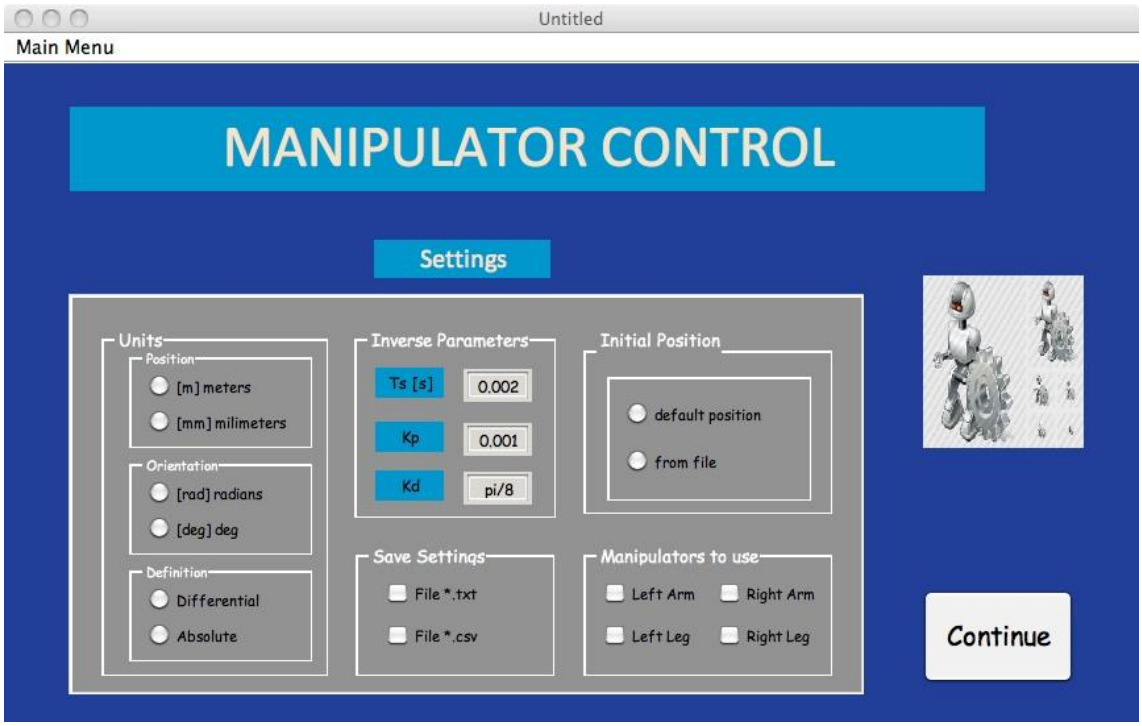



Figura 5.3.2.1: Pantalla inicial del subprograma Manipulator Control

En la figura 5.3.2.1 se puede ver que en primer lugar hay que indicar las unidades de posición (metros o milímetros) y las unidades de orientación (radianes o grados). La definición de las localizaciones podrá ser diferencial (Differential) o absoluta. La localización diferencial permite trabajar con incrementos de posición y orientación en caso de que no se tenga información en el espacio cartesiano con coordenadas absolutas respecto al sistema situado en el centro de masas.

El paso para la generación de la trayectoria es el tiempo Ts en segundos, y será el mismo para todos los manipuladores y trayectorias. Así mismo, se da la opción de controlar los parámetros de inversión del algoritmo de solución para realizar un estudio algo más detallado de este tipo de realimentaciones (Kp,Kd).
La configuración inicial de los motores del robot se puede elegir entre una configuración por defecto o una configuración guardada en archivo con extensión csv con el siguiente formato comentado en el apartado anterior de diseño de la interfaz.

Se podrán elegir los manipuladores a estudiar: brazo derecho, brazo izquierdo, pierna izquierda y/o pierna derecha. A su vez, se podrá elegir entre los manipuladores correspondientes a la pierna de apoyo y a la pierna flotante.

En el caso de la pierna soporte, la trayectoria generada será la del centro de masas. Esta circunstancia viene reflejada en el capitulo anterior de diseño de la interfaz.

### 5.3.3 Generación de trayectorias en el espacio de trabajo

Una vez validamos la opción anterior, el programa muestra una pantalla donde vienen tabulados los datos correspondientes a la trayectoria tales como los tiempos inicial y final de cada trayectoria, las coordenadas de posición iniciales y finales, el tipo de interpolación empleada en la posición, la orientación inicial y final (roll, pitch, yaw), etc.

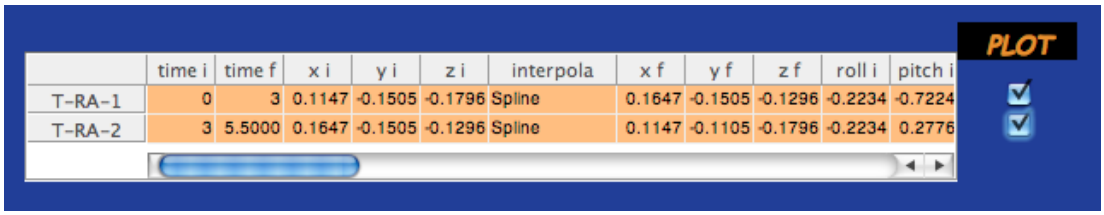



Figura 5.3.3.1: Panel de datos de las trayectorias

Además de proporcionar la opción de volver al menú principal, en esta pantalla podemos obtener una información más detallada del manipulador deseado pinchando en el menú correspondiente. En dicho menú podemos seleccionar la primera trayectoria a generar.

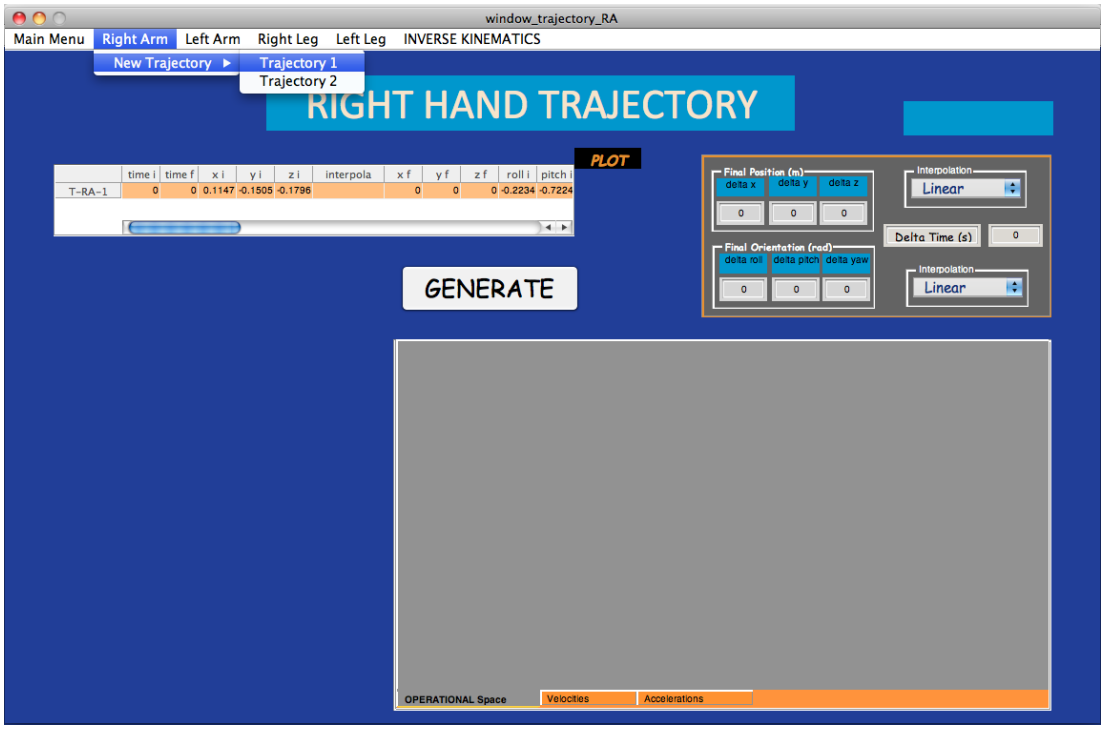



Figura 5.3.3.2: Menú desplegable de selección de trayectorias

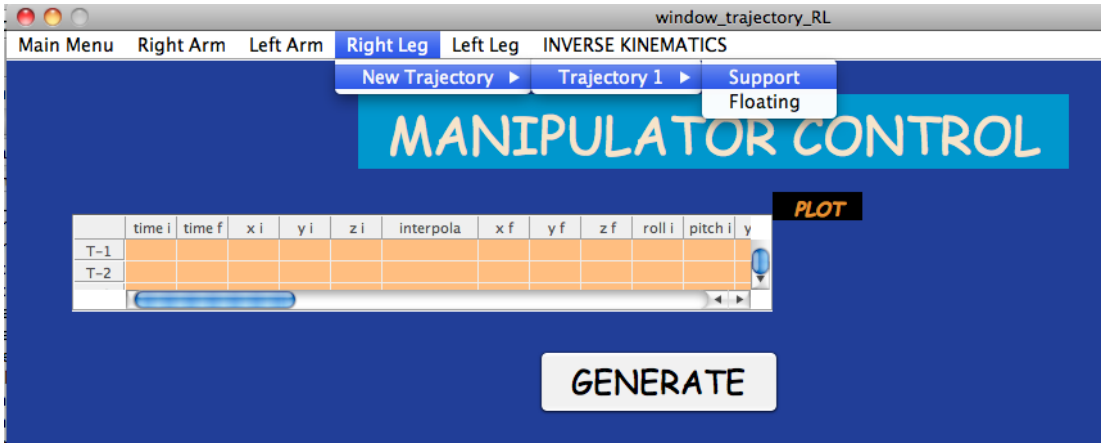



Figura 5.3.3.3: Menú desplegable de selección de trayectorias

En la figura de arriba se observa cómo en los menús de la pierna derecha e izquierda se puede elegir entre trayectoria con pie como soporte o con pie flotante. Por ejemplo si para la trayectoria 1 de la pierna derecha se elige como soporte, la trayectoria flotante quedará bloqueada en la siguiente trayectoria.

En la parte superior derecha se facilita un panel para que el usuario introduzca la posición y orientación final deseada, el tipo de interpolación a realizar, y la duración de la trayectoria en segundos.

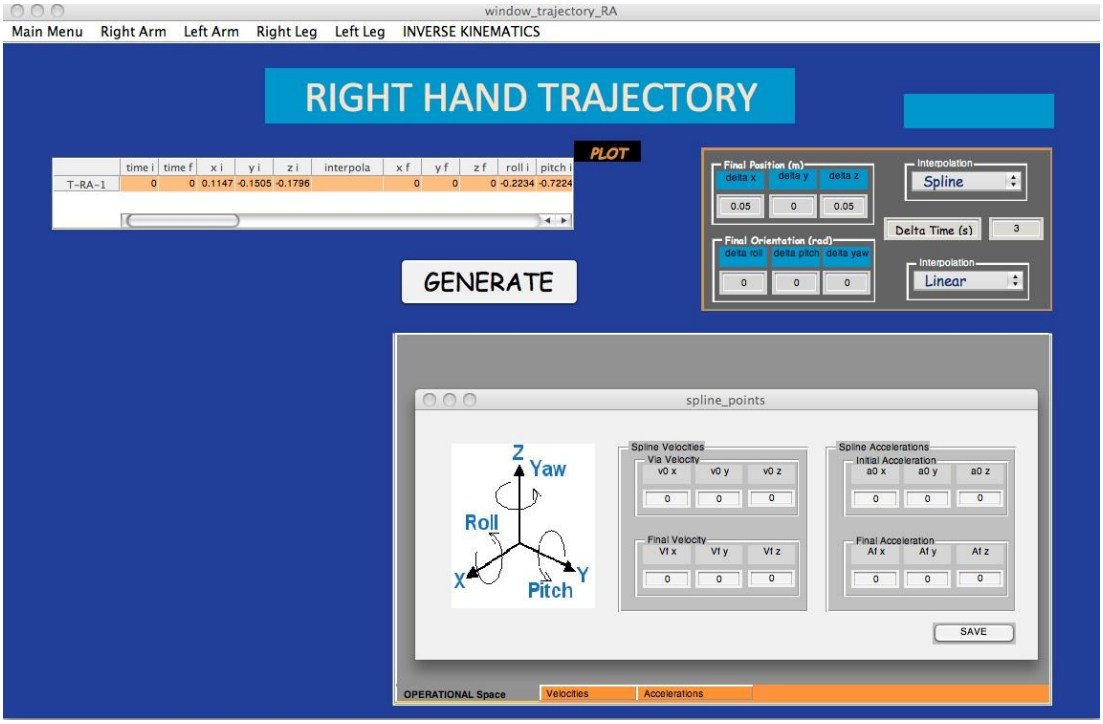



Figura 5.3.3.4: Panel de introducción de posición, orientación, tipo de interpolación y duración de la trayectoria

Una vez se pulsa el botón de generación de la trayectoria (GENERATE), dependiendo de el tipo de interpolación elegida, aparecen una ventana de inserción de velocidades y aceleraciones inicial y final.

Una vez introducidos todos los datos, el programa inicia el proceso de la generación de la trayectoria, y una vez finalizado aparece la opción de representar gráficamente dicha la evolución temporal de la posición (x,y,z) y la orientación (roll, pitch,yaw) en el espacio operacional así como en el espacio de velocidades y aceleraciones pinchando en la pestaña adecuada.

En el display de la zona inferior izquierda de la pantalla, se encuentra la representación gráfica en tres dimensiones del extremo del manipulador elegido.

En cada ventana del manipulador podemos generar una trayectoria adicional tomando como punto de partida la posición final de la trayectoria generada en primera instancia sin más pinchar en el menú del manipulador y eligiendo la opción de trayectoria 2 (Trajectory-2). Si representamos esta nueva trayectoria, aparecerán las curvas anteriormente comentadas con la salvedad de que el primer tramo corresponde a la trayectoria 1 (curva color azul) y de manera continua un segundo tramo correspondiente a la trayectoria 2 (curva color rosa).

Si deseamos aislar cualquiera de los dos tramos de la trayectoria final, podemos deseleccionar la casilla del check plot para que oculte la trayectoria no deseada.

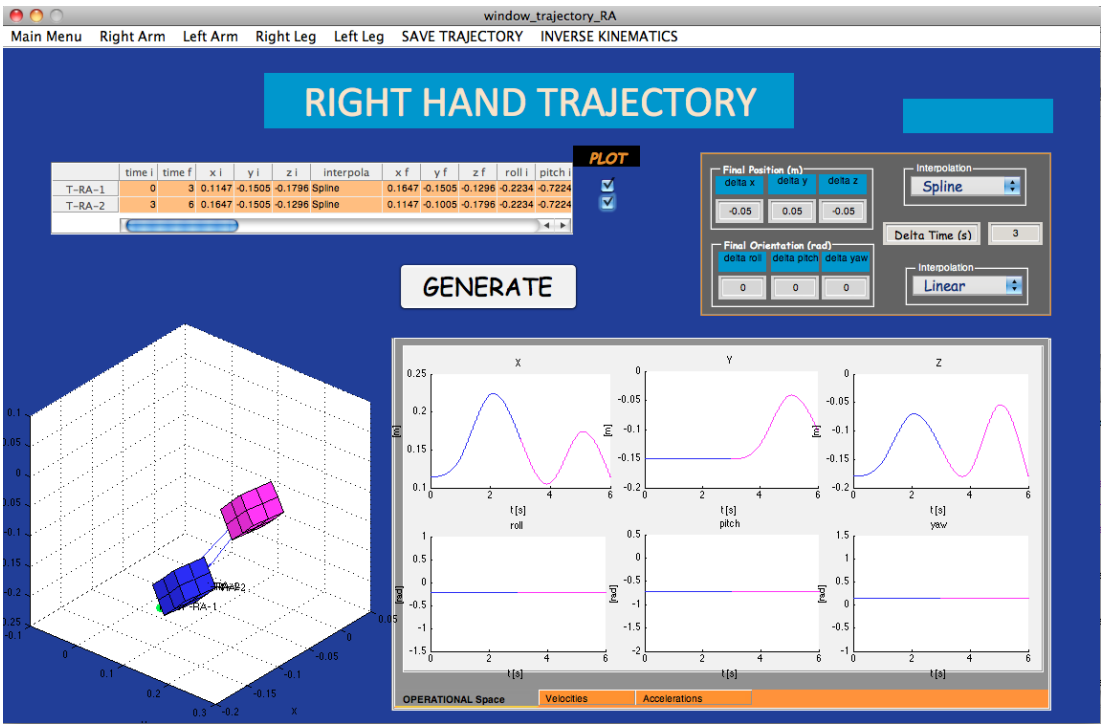



Figura 5.3.3.5: Ventana del manipulador seleccionado una vez generamos la trayectoria

Todas estas trayectorias se podrán salvar en un archivo csv o txt antes de pasar al menú de inversión cinemática.

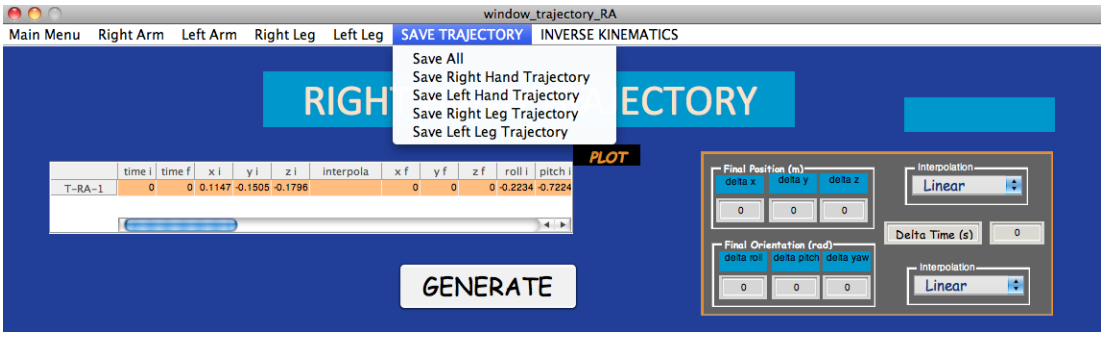



Figura 5.3.3.6: Menú para salvar las trayectorias generadas de cada manipulador

## 5.3.4 Inversión cinemática

En el menú que nos ocupa, el usuario puede modificar los parámetros del algoritmo de inversión pero en este caso no se puede modificiar el paso bien porque ya se han generado las trayectorias en el menú anterior, bien porque el usuario decide realizar la inversión cinemática a partir de un archivo csv ó txt que almacena una trayectoria anterior.

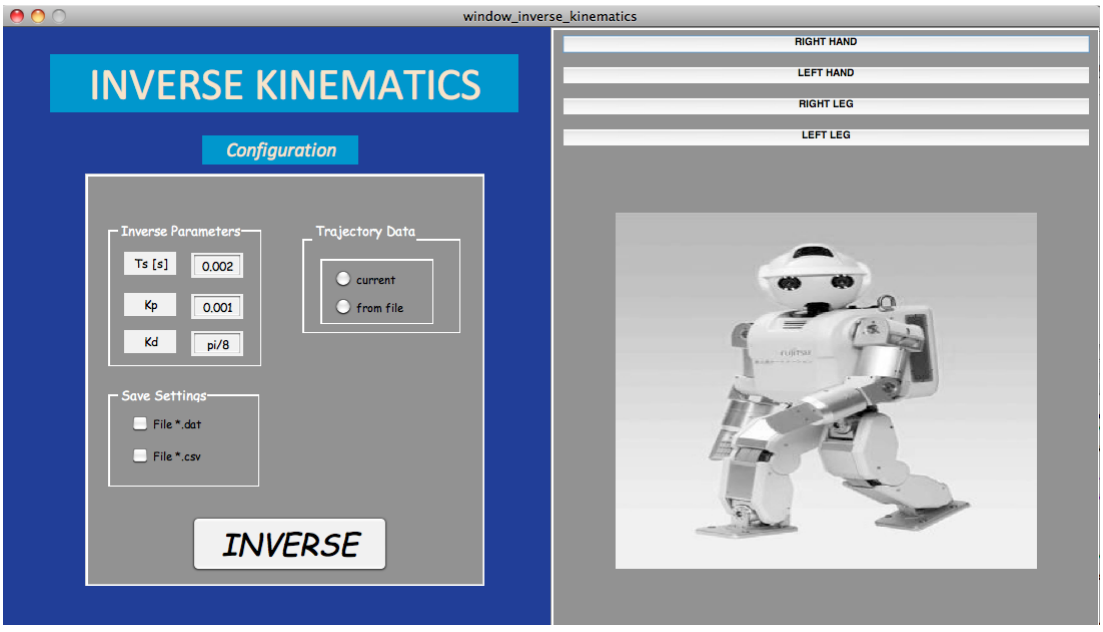



Figura 5.3.4.1: Aspecto del subprograma de inversión cinemática

Pulsando el botón de inversión (INVERSE) aparece una barra de espera mientras el programa realiza las operaciones de inversión. Una vez finalizado el proceso, se puede elegir una representación del espacio articular de cada

manipulador, de sus velocidades o de sus aceleraciones en el panel Graphic Options.

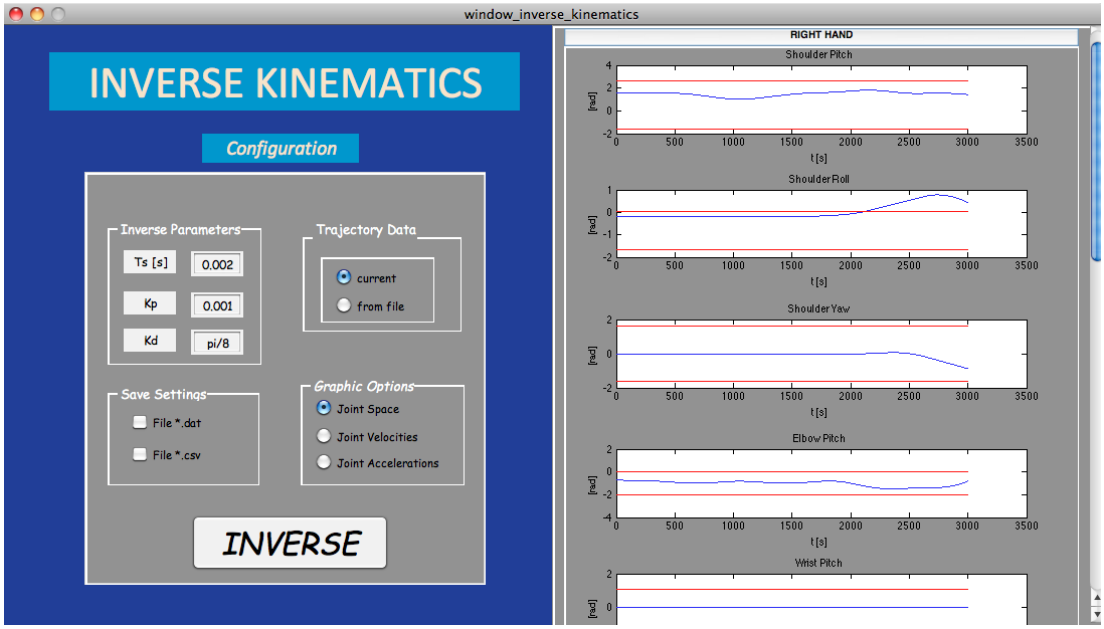



Figura 5.3.4.2: Vista del panel de representación de características del manipulador (Graphic Options)

Estas gráficas vienen recogidas en las pestañas pop-up situadas en la parte derecha de la pantalla para cada manipulador.

El principal interés de estas gráficas radica en la verificación de que nos encontramos dentro de los límites de giro de cada motor. Si las líneas representadas en la gráfica sobrepasan esos límites, indica que la trayectoria generada no puede ser realizada por la limitación operativa de los motores.

# 5.4 Steps Control

## 5.4.1 Introducción al control de paso

En este segundo bloque de la interfaz se va a generar un paso del robot. Cabe destacar que la generación del paso se hace en base a criterios de estabilidad estática del centro de masas, en ningún momento se tienen en cuenta criterios de la dinámica del robot.

Esta vez, el usuario tiene un control más restringido de los movimientos de los manipuladores, debido a las peculiares características que el movimiento de paso posee.

## 5.4.2 Definición de los parámetros de paso

En este apartado vamos a distiguir dos tipos de parámetros de paso. La vista de la pantalla inicial de este bloque de la interfaz se adjunta a continuación:


Figura 5.4.2.1: Aspecto de la pantalla incial del bloque STEP CONTROL de la interfaz

Tipos de soporte del paso:

**- Doble soporte:** es el movimiento del robot en el que ambos pies se encuentran fijos y que sirve de antesala del inicio del movimiento de simple apoyo.

Figura 5.4.2.2: Figura representativa del movimiento inicial de doble soporte

**- Simple soporte:** es aquel movimiento del robot manteniendo uno de los pies apoyado.

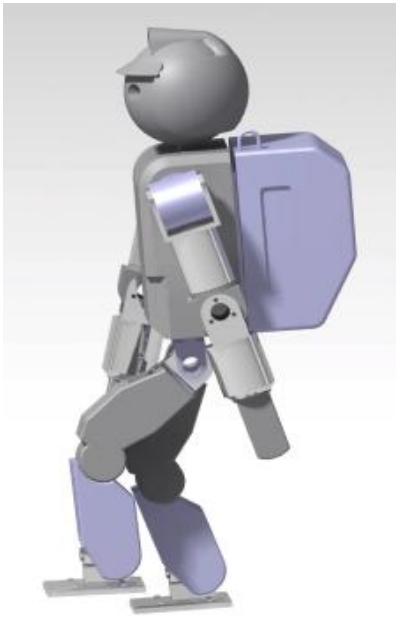


Figura 5.4.2.3: Figura representativa del movimiento inicial de simple soporte

El usuario podrá elegir qué pie hace de soporte, si el derecho o el izquierdo. Además tendrá la opción de hacer doble y simple soporte o solamente simple soporte.

Al igual que en la generación de trayectorias por manipuladores, el usuario puede elegir una posición inicial desde un archivo csv o una por defecto proporcionada por el programa.

Las variables a introducir son:

**- Longitud del paso:** por defecto esta distancia se reparte por igual entre la aportación del movimiento del centro de masas y del desplazamiento del pie flotante.

**- Altura del paso:** la altura de paso se considera el punto final de la fase 1 de simple soporte y el punto de inicio de la fase 2 de simple soporte cuyo punto final es el suelo.

**- Duración del movimiento total**

**- Tiempo de paso Ts**

## 5.4.3 Diseño del paso

Con el fin de clarificar las opciones que la interfaz ofrece, el usuario dispone de notas aclaratorias en la parte superior de cada panel.

En esta ventana podemos encontrar un panel asociado a cada movimiento a estudiar, es decir, simple soporte, doble soporte y movimiento de brazos.

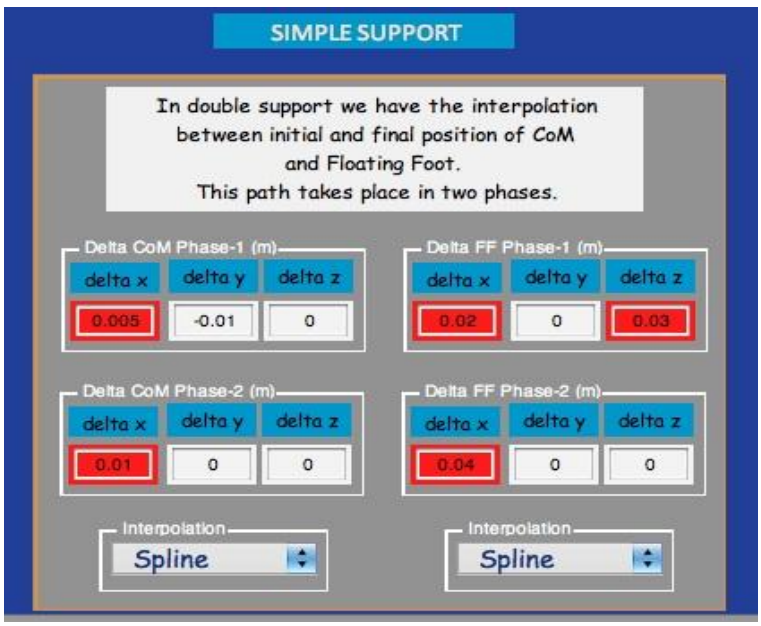



Figura 5.4.3.1: Panel de introducción de datos del movimiento de simple soporte

Las celdas en rojo no podrán ser modificas por el usuario debido a que están basadas en las mínimas restricciones de estabilidad que el robot precisa para realizar un paso en condiciones de seguridad.

En cada uno de estos movimientos se tiene la posibilidad de elegir el tipo de interpolación para cada movimiento, pudiéndose combinar interpolaciones distintas en un mismo panel, por ejemplo en el panel de simple soporte, se puede usar una interpolación lineal para la aportación en el movimiento del centro de masas y una interpolación spline para la parte de movimiento correspondiente al pie flotante.

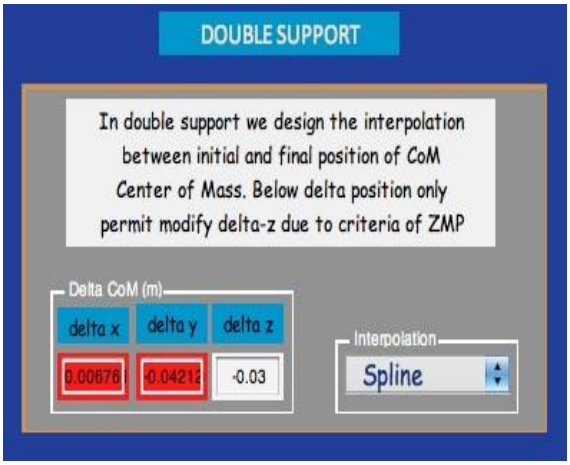



Figura 5.4.3.2: Panel de introducción de datos del movimiento de doble soporte

Notar que en el caso de doble soporte no se va a poder combinar entre dos tipos de interpolaciones distintas porque solo se genera la trayectoria de un punto, el centro de masas.

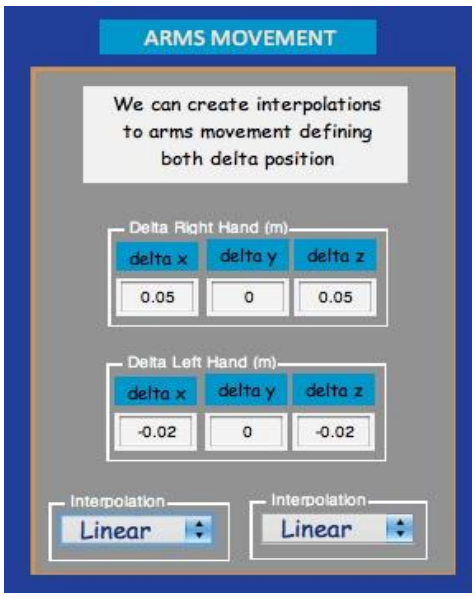



Figura 5.4.3.3: Panel de introducción de datos del movimiento de brazos

Una termine la computación del paso generado tras pulsar el botón GENERATE STEP, automáticamente aparece una simulación en tres dimensiones del movimiento de ambos pies y el centro de masas en la pestaña CoM (Center of Mass)

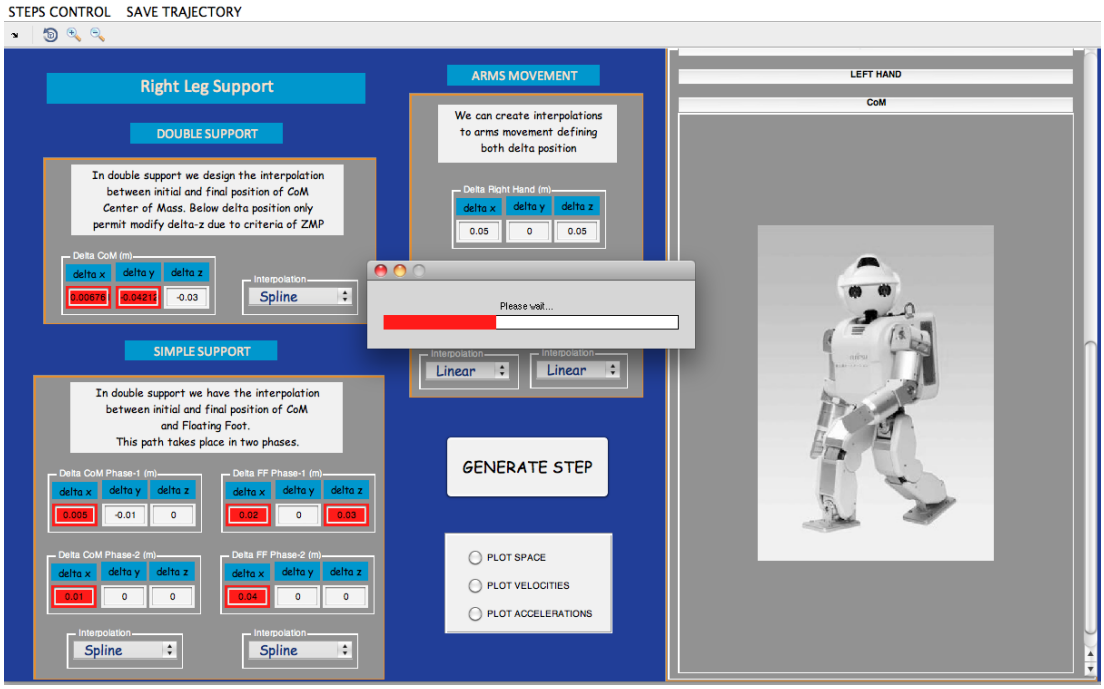



Figura 5.4.3.4: Pantalla de computación del movimiento de paso seleccionado

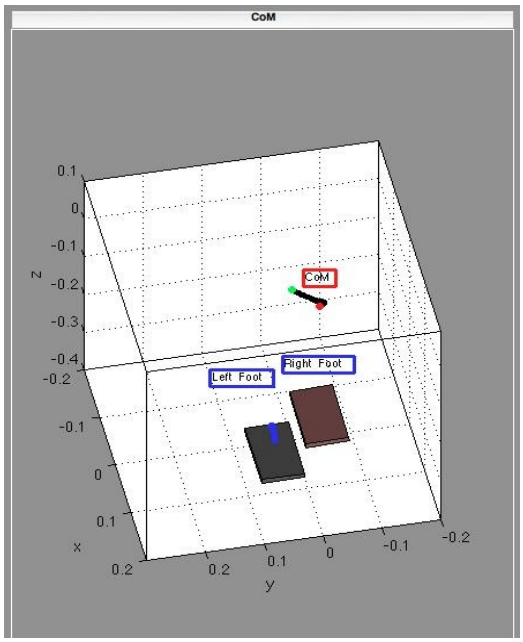



Figura 5.4.3.5: Simulación en tres dimensiones del paso generado

De nuevo se da la posibilidad de representar gráficamente el espacio de articulaciones, las velocidades o las aceleraciones, tal y como se hacía en Manipulator Control.

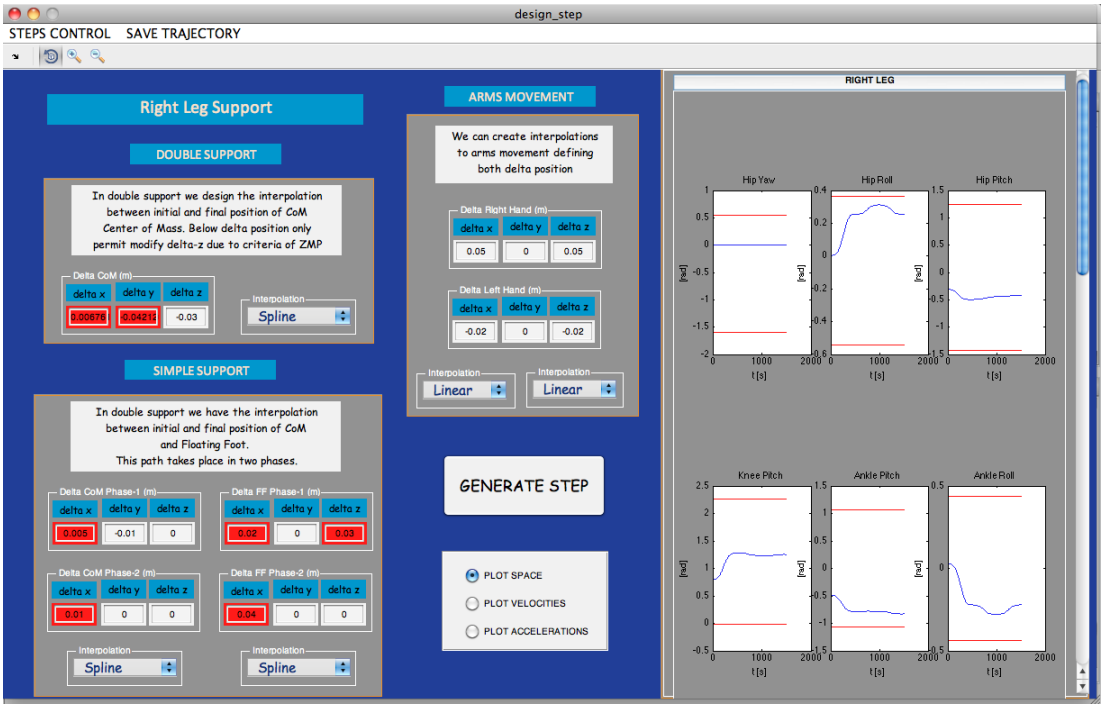



Figura 5.4.3.6: Representación del espacio de articulaciones, velocidad y aceleración

Por último, en el menú SAVE TRAJECTORY se puede grabar los resultados de la inversión cinemática en archivos *.txt o *.csv para un posterior uso en software de simulación, en excel o en Matlab. También cabe la posibilidad de almacenar todos los resultados en una variable *.mat de tipo estructura de datos para uso interno del software Matlab.

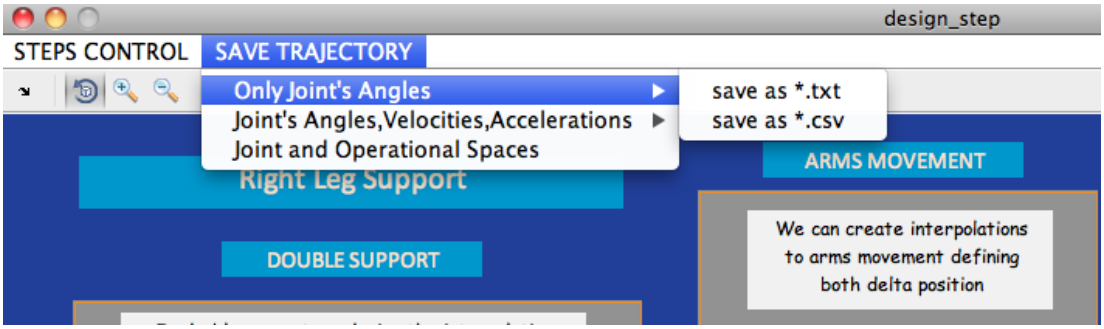



Figura 5.4.3.7: Menú SAVE del bloque STEP CONTROL

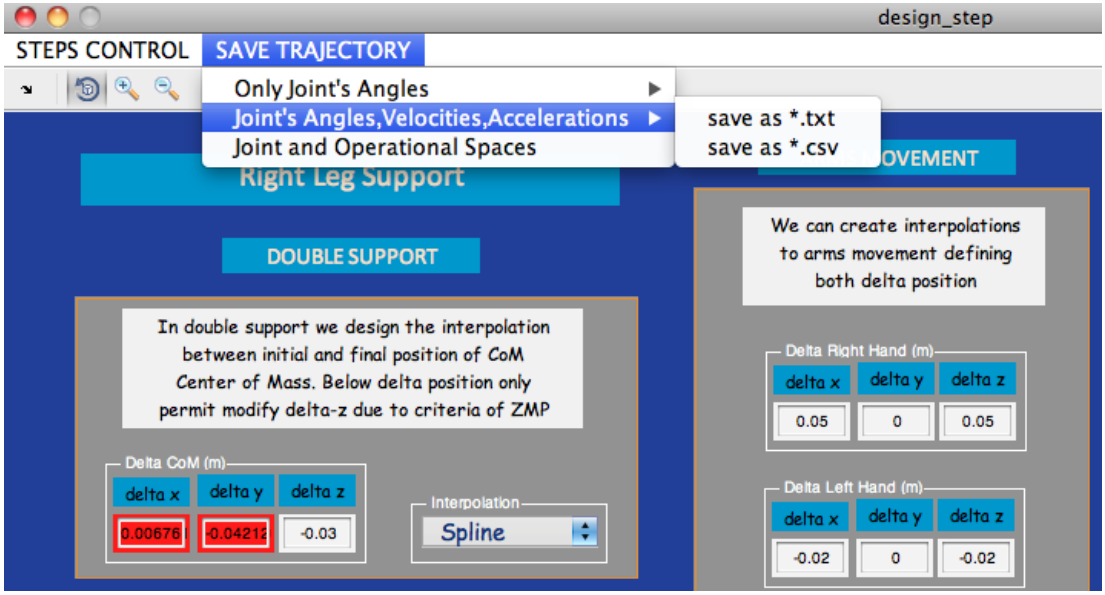



Figura 5.4.3.8: Menú SAVE del bloque STEP CONTROL

# 5.5 Verificación de resultados obtenidos en la interfaz con CATIA V5

## 5.5.1 Justificación

A la hora de simular los movimientos del robot HOAP-3 con la interfaz de MatLab, se pueden encontrar razonables limitaciones en capacidad visual de representación en 3D que dicho programa ofrece.

Con el fin de mostrar una representación más fidedigna de los movimientos del robot, se ha elaborado un modelo en 3D del mismo mediante el programa CATIA V5 de Dassault Systems.

Este modelo permitirá verificar visualmente la adecuación de algunos resultados obtenidos con la interfaz anteriormente descrita, especialmente los relativos al movimiento de brazos. En este caso, el extremo final de los manipuladores brazo derecho y brazo izquierdo, están referenciados al centro de masas, considerado como sistema inercial. En el modelo realizado esto no supone ningún problema, dado que la herencia de los objetos gráficos permitirá representar de forma adecuada el movimiento de cada articulación de los brazos.

Para el caso de las piernas, sin embargo, el modelo tiene que considerar cuál es el pie soporte y fijarlo como referencia para el resto de objetos que constituyen el robot. Esto supone una dificultad puesto que en la simulación de cada paso hay que modificar el modelo realizado en CATIA.

## 5.5.2 Metodología empleada

**- Creación de las piezas:**

Al construir el modelo, se ha tomado como punto de partida tanto del esquema con las joints como del aspecto visual recogidos en el manual del HOAP-3 proporcionado por fabricante Fujitsu.

Para poder simularlo como una cadena cinemática, el robot debe ser construido como un conjunto de piezas inicialmente independientes entre sí, pero que conservan las dimensiones reflejadas en el manual. Así, se puede dar a cada pieza su correspondiente material y propiedades para luego ensamblarla en el conjunto final que será el robot. Este proceso se realiza mediante el módulo de Part Design de CATIA.

En la siguiente figura se muestra el aspecto del software mencionado en el proceso de creación de una pieza, en este caso en pie derecho del robot.

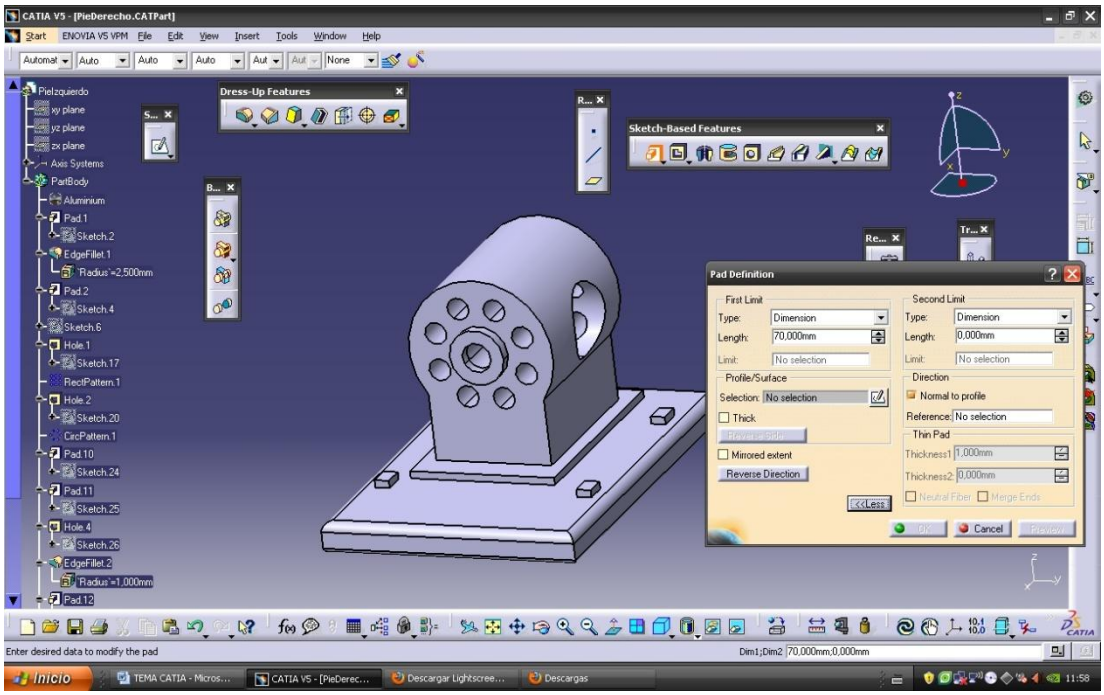



Figura 5.5.2.1: Proceso de creación de las piezas del robot

**- Ensamblaje del modelo:**

Una vez construidas todas las piezas que componen el robot, se ejecuta el módulo Assembly de CATIA, donde se puede introducir los grados de libertad del robot mediante restricciones de diversos tipos.

En el este caso, los parámetros de simulación van a ser principalmente ángulos de giro de los motores que componen las articulaciones. El diseño del robot en CATIA se ha hecho expresamente para que todos los contactos sean de revolución y así poder simularlo con los datos obtenidos mediante la interfaz de MatLab.

En las siguientes figuras podemos observar el punto inicial y final del proceso de ensamblaje del robot HOAP-3.

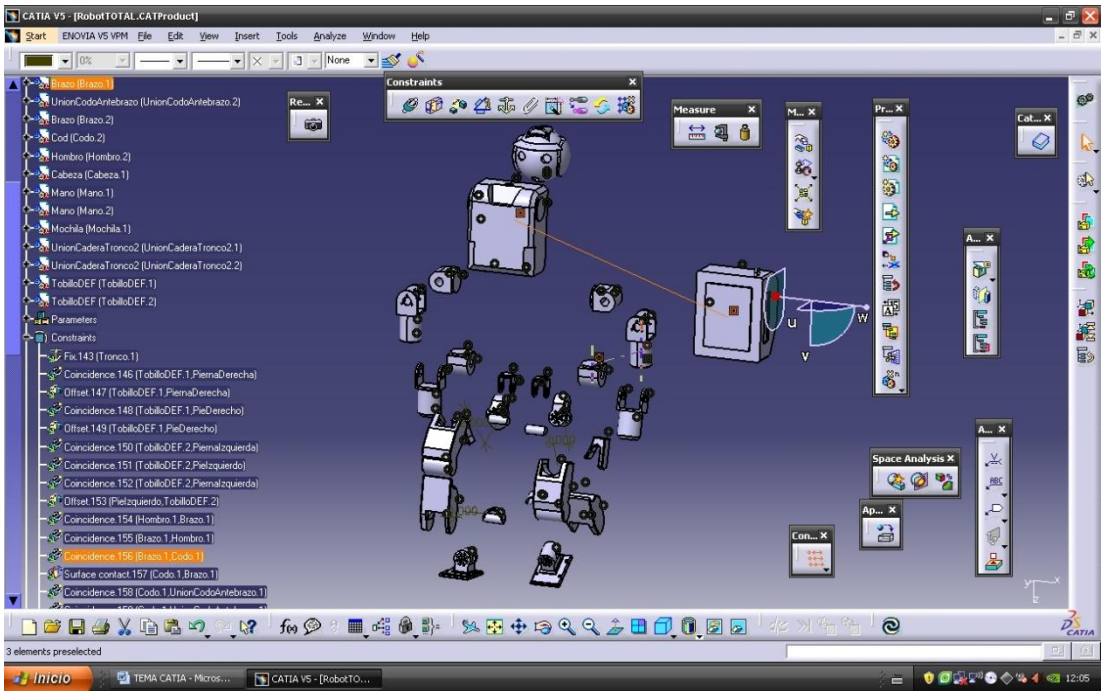



Figura 5.5.2.2: Piezas del robot antes de ser ensambladas

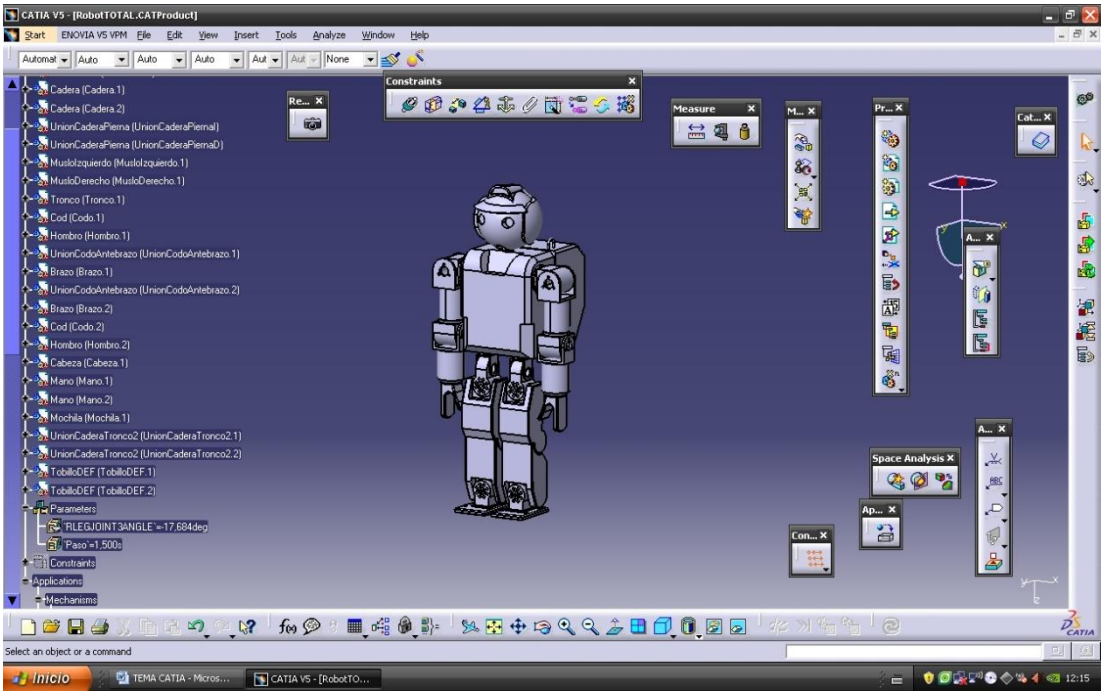



Figura 5.5.2.3: Robot una vez ensambladas todas las piezas

**- Parametrización con comandos:**

Terminado el ensamblaje, mediante la operación *Assembly Constrains Conversion* del módulo DMU Kinematics de CATIA, se convierten las restricciones en juntas de revolución para todas las articulaciones.

Como al fin y al cabo se trata de una cadena cinemática, hay que fijar una de las piezas para que el resto de la cadena pueda moverse con una referencia. Se ha tratado de simular el movimiento de andar en el robot HOAP-3, por lo que se considera que la mejor forma de visualizar dicho movimiento es fijando primero un pie del robot (que servirá de punto para que robot dé un primer paso) y una vez ha completado el paso, será el otro pie el que quede fijo, sirviendo en este caso de punto de apoyo del siguiente. Y así sucesivamente.

Para poder introducir los valores calculados en la interfaz, parámetros de giro asociados al movimiento de revolución de las articulaciones, una vez que se ha realizado este paso, CATIA indica que el mecanismo ya puede ser simulado.

Se ha de asegurar una simulación lo más acorde a la realidad posible y para ello, se ha introducido los valores máximos y mínimos de giro de los motores de las articulaciones. Además, en el display de control de simulación creado en CATIA, se puede asignar el paso angular (en grados) que se desee para el modelo, esto es, cada cuantos grados va a representar la trayectoria circular de los movimientos de giro de los motores. En general, se ha asignado un valor de 5º, pero puede ser modificado a placer (mini pantalla en la parte inferior derecha de la figura 5.5.2.4)

El aspecto de la interfaz de simulación en CATIA tiene el siguiente aspecto:

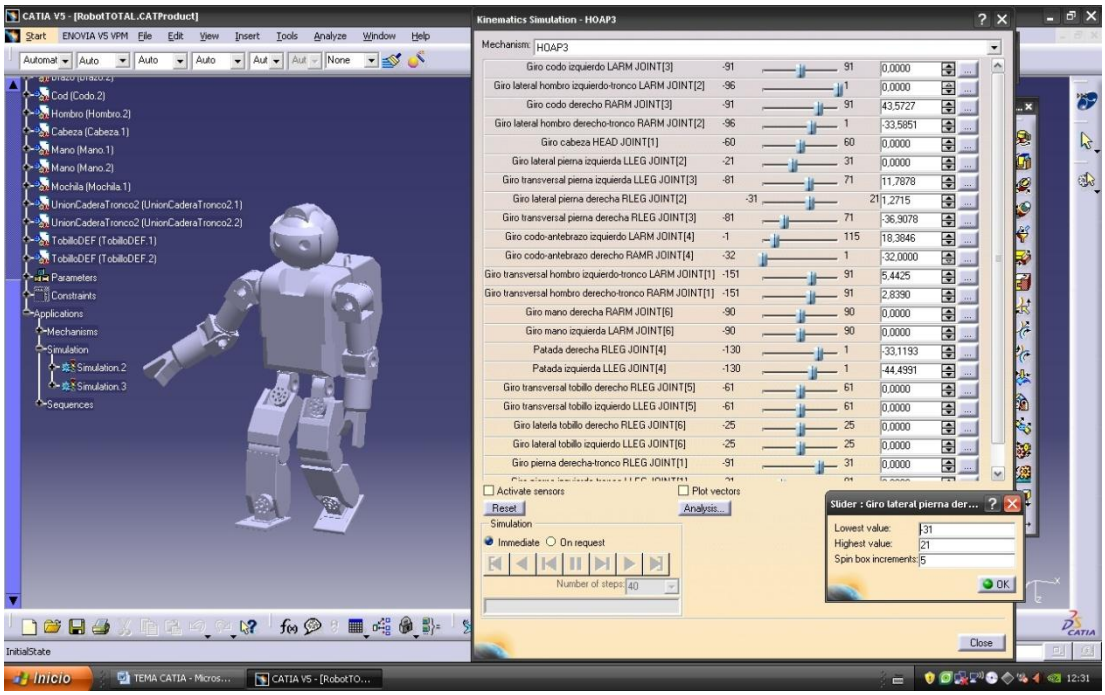



Figura 5.5.2.4: Interfaz de CATIA para simular los movimientos del robot HOAP-3

**- Simulación de los movimientos del HOAP-3:**

Una vez preparada la simulación con comandos, se toman los valores angulares tabulados de los ángulos de giro de los motores y se observa donde hay cambios de sentido en el giro de las articulaciones.

En esos cambios de sentido angular, con sus respectivos tiempos asociados, se darán los puntos límite de final (o inicio) de movimiento de cada articulación. Introduciendo cada uno de ellos y asignándole el tiempo adecuado, se crean simulaciones de movimientos que luego se montan en una secuencia para poder crear un vídeo de un movimiento completo.

Todo esto se realiza en el mismo módulo DMU Kinematics de CATIA mediante las operaciones de *Simulations with commands* y *Edit Sequence*.

Cabe destacar que el punto de partida de movimientos es la posición erguida del robot tal y como viene representado en la figura 5.5.2.3.

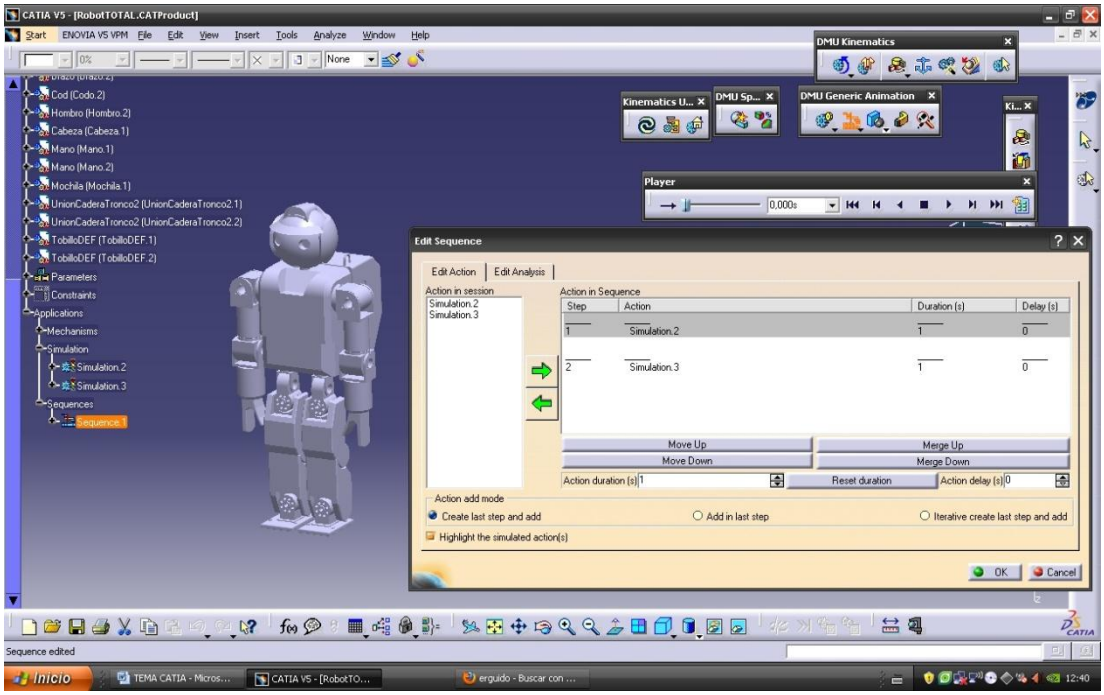



Figura 5.5.2.5: Montaje de los movimientos simulados

Una vez montada la secuencia de movimientos que forman una acción (caminar, levantar un brazo y girarlo, etc…) se exporta el vídeo a los formatos habituales de trabajo, en este caso, [*.avi].

Dado el alto detalle constructivo del modelo, estos archivos ocupan un espacio importante, por lo que conviene no hacer simulaciones de cadenas de movimientos muy largas, o si se hacen, editarlas posteriormente en un programa de edición de vídeo para comprimirlas de algún modo.

### 5.5.3. Ventajas de simular algunos movimientos en CATIA V5

La ventaja principal de simular los movimientos con CATIA es la oportunidad de observar con mucha claridad las trayectorias que el robot realiza con los parámetros que se han introducido. No obstante, la matemática que se haya en el cálculo de esos valores no puede ser obtenida en CATIA con la precisión con la que se ha obtenido en MatLab por lo que un programa complementa al otro.

El paquete CATIA ofrecido por Dassault Systems, permite un abanico muy amplio de opciones de análisis del sistema, tales como:

- Análisis de interferencias y colisiones entre piezas
- Análisis dinámico e inercial del modelo
- Creación de superficies de barrido producto de los movimientos del robot
- Etc.

Para hacer un uso adecuado de algunas de las herramientas mencionadas, se debería introducir los parámetros de las piezas del robot (peso, geometría, densidad…) con extremada precisión, de lo contrario el análisis iría propagando los errores llevándose a resultados erróneos. Lamentablemente no se dispone de dicha información por lo que se deja esta vía como una posible línea de investigación para futuros estudios.

# Capítulo 6

# Estudio económico del proyecto

# 6.1 Planificación por etapas

Para la realización de este proyecto, al igual que ocurriría con cualquier otra empresa de magnitud semejante, ha sido necesaria una planificación previa. El objetivo de esta planificación ha sido realizar el reparto de recursos para las tareas que se han de realizar de acuerdo al diseño elegido, así como estimar el tiempo que dichas tareas necesitarán para su realización con los recursos asignados. Esto es muy importante a la hora de calcular y distribuir los recursos necesarios para su realización.

Gracias a la planificación, se ha tenido un control suficiente sobre los plazos temporales y materiales en el proyecto, lo que ha permitido su realización dentro de un margen de tiempo adecuado y sin necesidad de hacer acopio de recursos materiales o personales de última hora.

A continuación se muestra la planificación temporal que se ha seguido para la realización de la interfaz de usuario de generación de trayectorias.

| | | |
|---|---|---|
| **Preparación del proyecto** | 18 días | Fase de análisis |
| Estudio preliminar de los requisitos | 5 dias | |
| Estudio de las tecnologías para el desarrollo | 3 dias | |
| Estudio de los proyectos relacionados | 5 dias | |
| Estudio de las utilidades disponibles | 1 dia | |
| Estudio de las posibles vías de desarrollo | 3 dias | Fase de diseño |
| Instalación del SW para el desarrollo | 1 dia | |
| **Integración y Modificación de los algoritmos cinemáticos** | 4 días | Fase de desarrollo |
| **Desarrollo de la interfaz** | 30 días | |
| Desarrollo modular | 20 dias | |
| Desarrollo y depuración de la interfaz | 10 días | |
| **Evaluación del funcionamiento** | 1 semana | Fase de evaluación |
| **Desarrollo de la memoria del proyecto y la documentación relacionada (Fase de documentación)** | 2 semanas | Fase de documentación |

Tabla 6.1.1: Descomponsición de las tareas

Esta duración se refiere únicamente al tiempo dedicado a cada tarea, y no al tiempo total desde que una tarea comienza hasta que es finalizada, ya que este tiempo ha sido mayor puesto que la disponibilidad para la realización del proyecto no ha sido exclusiva, sino que ha debido ser compaginada con el trabajo en la empresa.

Por otro lado, debido a la escasez de recursos humanos disponibles, estas tareas han debido competir por el uso de los mismos, dando lugar a un tiempo total de desarrollo que podría haberse reducido sustancialmente en caso de haber dispuesto de una cantidad de recursos mayor. Si el número de recursos materiales y

humanos hubiera sido mayor, algunas de las tareas habrían podido realizarse en paralelo, reduciendo de esta forma el tiempo de desarrollo global del proyecto.

No obstante, a la vista de los resultados, dado que no fue necesario cumplir con unos requisitos temporales estrictos para la entrega del proyecto, los recursos de los que se ha dispuesto para este proyecto han demostrado ser suficientes.

# 6.2 Presupuesto

A la hora de estudiar la viabilidad de cualquier proyecto, el presupuesto es una herramienta fundamental a tener en cuenta. Este proyecto no ha sido una excepción. Pese a que se trata de una herramienta inherentemente imprecisa, ya que para realizar presupuestos con precisión sería necesario disponer de información sobre el desarrollo y sobre posibles eventualidades que, en la inmensa mayoría de los casos, al comienzo de un proyecto sencillamente no está disponible, el presupuesto es clave para realizar una estimación bajo supuesto de que las fechas y condiciones que se estiman en la planificación son las correctas.

Lógicamente, cualquier variación de los supuestos iniciales a lo largo del desarrollo hace necesaria una revisión del presupuesto.

En cuanto a los perfiles de personal necesarios para el desarrollo del proyecto, éstos han sido los siguientes:

- Personal encargado de las tareas de documentación.
- Programador.
- Jefe de proyecto.

No obstante, sólo ha habido una persona física para el desarrollo de todo el trabajo, la cual ha desempeñado los papeles que corresponden a cada uno de los perfiles anteriores. Se ha considerado además necesaria la presencia de un jefe de proyecto que coordine y supervise las tareas que han de llevarse a cabo, sugerir líneas de actuación para la resolución de problemas y trabajar activamente en la revisión y corrección tanto de aspectos técnicos como de documentación.

En cuanto al ordenador personal utilizado para el desarrollo del software, se trata de un ordenador portátil MacBook Pro basado en un Intel Core 2 Duo a 2,53 GHz, con 4 Gb de memoria 1067 MHz DDR3 y 250 Gb de disco duro. Las prestaciones de este equipo cumplieron con las expectativas para el desarrollo. Para la compilación y creación del ejecutable se ha utilizado otro ordenador portátil marca Dell, basado en un Intel Core 2 con 2 Gb de memoria.

Aquellas herramientas propietarias utilizadas para la documentación (Microsoft Office), desarrollo de la interfaz (Matlab), así como el sistema operativo (Windows XP Professional SP2 y MAC OS X), deben ser tenidas en cuenta.

Vistos los recursos necesarios a incluir en el presupuesto, podemos pasar a realizar las mediciones. Con respecto a las mediciones, cabe decir que se trata de una estimación de los recursos necesarios para el desarrollo del proyecto, ya sean éstos materiales o humanos. En el caso de los recursos materiales, es necesario indicar el número de unidades necesarias de cada elemento. Por otro lado, con respecto a los recursos humanos, es necesario indicar el número de horas de trabajo que cada persona implicada en el proyecto debe dedicar. Las mediciones para el presente proyecto se muestran a continuación:

| Recurso humano | Horas de trabajo | Precio (€/h) |
|---|---|---|
| Programador | 300 | 9,8 |
| Jefe de proyecto | 200 | 12,79 |
| Persona encargada de la documentación | 200 | 9,8 |
| **Recurso material** | **Unidades** | **Precio/unidad** |
| Ordenador portátil | 1 | 800€ |
| Ordenador Mac | 1 | 1.200€ |
| Software Microsoft Windows | 1 | 60 |
| Software Microsoft Office 2008 | 1 | 170 |
| MATLAB R2011a | 1 | aportado por el cliente |
| CATIA | 1 | aportado por el cliente |

Tabla 6.2.1: Costes y recursos asociados al proyecto

Los costes asociados a estas horas de trabajo se han obtenido de acuerdo al convenio colectivo nacional de empresas de ingeniería y oficinas de estudios técnicos, donde se establece, en la revisión salarial del 28 de Enero de 2010, que las tareas desarrolladas por profesionales perteneciente a los grupos 26: I (jefe de proyecto), IIB (programador y personal encargado de la documentación) tienen los siguientes niveles retributivos:

| Grupo | Nivel retributivo | Sueldo base mensual(€) |
|---|---|---|
| I | 1 | 2046,96 |
| II A | 2 | 1698,54 |
| II B | 3 | 1567,89 |
| III B | 5 | 1306,58 |

Tabla 6.2.2: Niveles retributivos de acuerdo a cada grupo profesional

En la tabla anterior, como podemos ver, se muestran además los precios unitarios, los cuales muestran el coste por hora de trabajo de cada empleado en el caso de recursos humanos. Para los recursos materiales, muestran el precio de todos los elementos utilizados en el proyecto.

Teniendo en cuenta todos los elementos vistos hasta ahora, se puede estar en condiciones de elaborar un presupuesto general del proyecto para realizar una valoración definitiva de su coste:

| Recurso humano | Precio (€/h) | Horas de trabajo | Coste del recurso (€) |
|---|---|---|---|
| Programador | 9,8 | 300 | 2940 |
| Jefe de proyecto | 12,8 | 200 | 2560 |
| Persona encargada de la documentación | 9,8 | 200 | 1960 |
| **Recurso material** | **Precio/unidad (€)** | **Unidades** | **Coste del recurso (€)** |
| Ordenador portátil | 800 | 1 | 40 |
| Ordenador Mac | 1200 | 1 | 60 |
| Software Microsoft Windows | 60 | 1 | 3 |
| Software Microsoft Office 2008 | 170 | 1 | 8,5 |
| MATLAB R2011a | Cliente | 1 | 0 |
| CATIA | Cliente | 1 | 0 |
| Coste total (€) (supuesto sin CATIA) | Supuesto-1 | | **7571,5** |

| | | | |
|---|---|---|---|
| Precio de venta (€) 20% Beneficio | Supuesto-1 | | 9085,8 |
| | | | |

Tabla 6.2.3: Presupuesto total (sin CATIA)

El presupuesto total de este proyecto asciende a la cantidad de 9086 €

Leganés a 20 de Febrero de 2012

El ingeniero proyectista

Fdo. Daniel Juan García-Cano Locatelli

Se ha aplicado la siguiente fórmula para el cálculo de la amortización de estos recursos:

*A·C·D/B = Amortización* , donde *A* es el número de meses desde la fecha de facturación en que el equipo es utilizado; B es el período de depreciación (60 meses), C es el coste del equipo (sin IV A), y D es el porcentaje de uso que se dedica al proyecto del recurso en cuestión.

# Capítulo 7

# Futuras líneas investigación

# 7.1 Introducción

El presente proyecto ha intentado cubrir de la mejor manera posible los objetivos que al principio se planteaban. Como en todo proyecto, el "deadline" impone un tiempo limitado para realizar el trabajo en toda su extensión, no se ha podido profundizar en algunas de las facetas y áreas que la interfaz de usuario ha generado y que se consideran importantes para tenerse en cuenta en futuros proyectos fin de carrera.

# 7.2 Generación de caminata

Tras el diseño de un paso en Step Control, el siguiente avance sería poder generar una caminata. El usuario introduciría la longitud a recorrer y el tiempo que quiere invertir en dicha caminata. La interfaz debería dar una primera respuesta calculando previamente si es posible o no realizar la caminata en el tiempo que desea el usuario teniendo en cuenta las restricciones cinemáticas impuestas en el robot.

Dado que la interfaz que se ha creado en este proyecto se basa exclusivamente en criterios cinemáticos, sería interesante incorporar el modelo dinámico del robot y utilizarlo en los cálculos y simulaciones de la caminata.

Como se verá en el próximo capítulo de Conclusiones y resultados, se ha podido disponer de un modelo dinámico creado por Paolo Pierro para comprobar ciertos movimientos en la generación de un paso del robot HOAP-3.

## 7.2.1 Aspectos a tener en cuenta en la generación de una caminata

En la generación de una caminata, off-line, se ejecutará el código de resolución de inversión cinemática, que se aporta en el presente proyecto, en cada paso generado. De tal forma que cada paso tenga como configuración articular inicial la solución de la inversión cinemática del paso anterior. Podría parecer sencillo este bucle pero hay que tener en cuenta algún criterio de estabilidad estática para garantizar una seguridad mínima en la caminata.

Este criterio se puede basar en la proyección del centro de masas en el plano X-Y, si este punto proyectado se encuentra dentro del póligono de sustentación formado por ambos pies se puede garantizar una cierta estabilidad estática.

# 7.3 Workspace

El espacio de trabajo del robot, el espacio que puede alcanzar cada extremo final de los manipuladores del humanoide, es una faceta interesante que pudiera albergar la interfaz de usuario de generación de trayectorias.

Si dada una localización inicial del extremo de un manipulador dado, se quisiera alcanzar una localización final, se podría saber antes de realizar la inversión cinemática si esa localización es alcanzable o no en un primer paso o si es necesario un desplazamiento previo para alcanzar tal meta.

Analíticamente, la función que describe este espacio alcanzable de un manipulador se complica a medida que aumenta el número de grados de libertad de dicho manipulador. Como primer trabajo se podría hallar esta función para el brazo derecho e izquierdo. Pero como se ha dicho antes, analíticamente puede ser complicado hallar esta función, se plantea hallar este espacio de otra manera.

Como se parte de la cinemática directa resuelta del robot, se podría crear un espacio discreto de soluciones dentro del intervalo de movimiento de giro de cada articulación y con un paso angular de un grado o medio grado se ejecuta un bucle de cinemática directa para todos esos intervalos, generando una nube de puntos en el espacio.

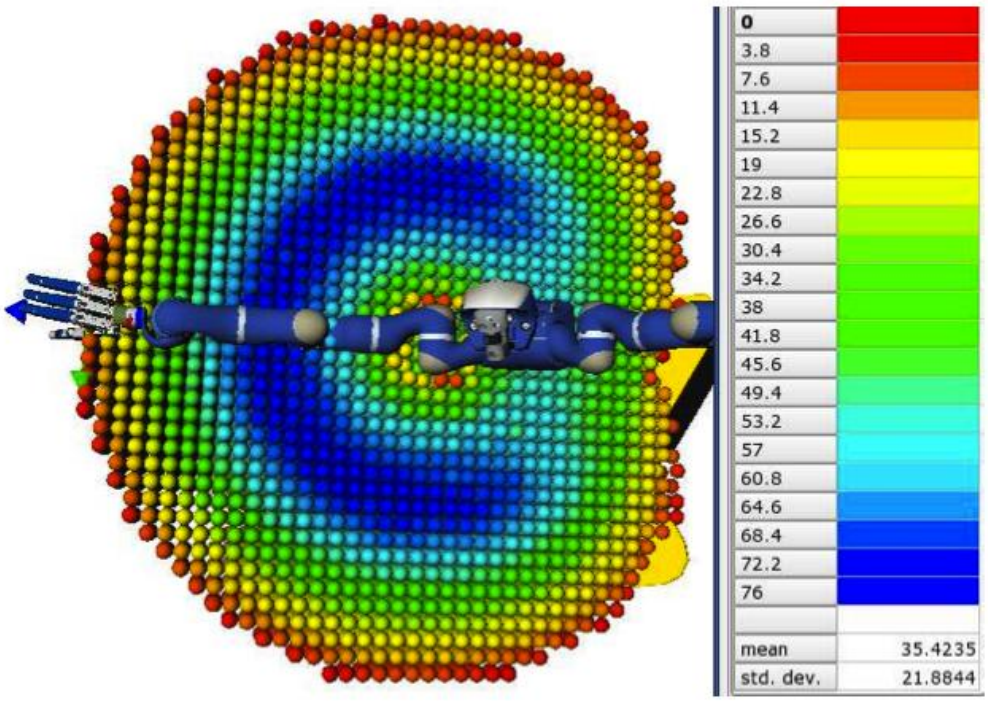



Figura 7.3.1: Nube de puntos del espacio de trabajo del robot

El análisis del espacio de trabajo de un robot también es un paso fundamental a la hora de exigir buenas prestaciones en el diseño mecánico del mismo. Por lo tanto, es necesario realizar un estudio detallado de los parámetros geométricos y calcular sus valores óptimos para obtener los mejores resultados en el diseño.

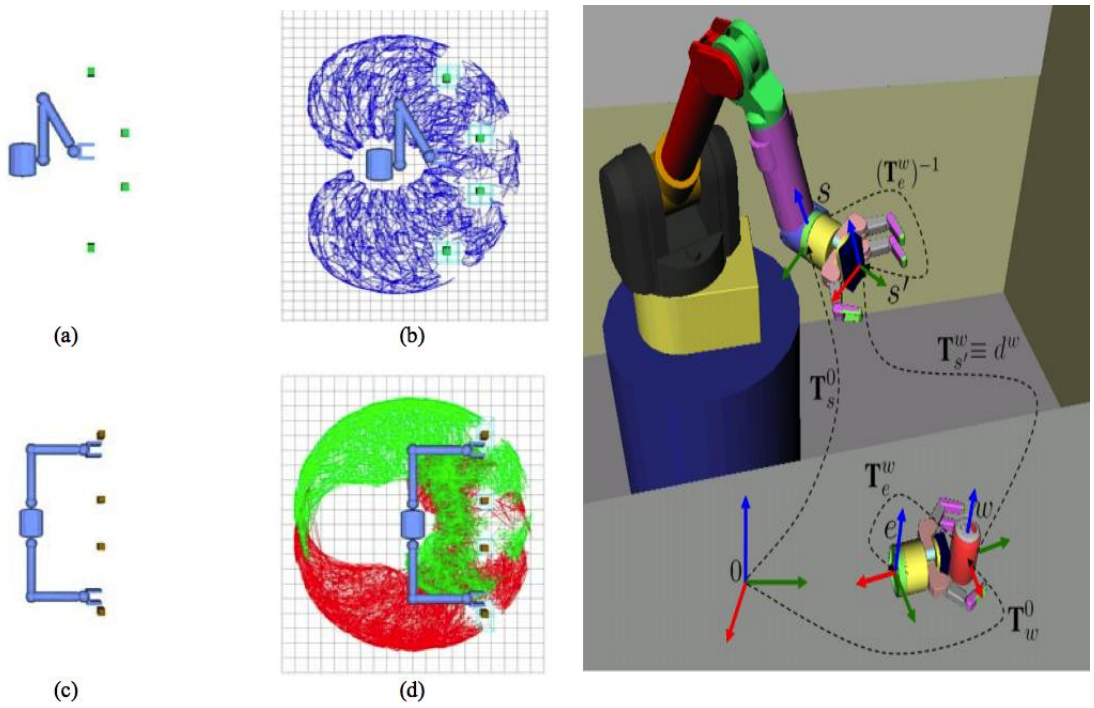



Figura 7.3.2: Análisis del espacio de trabajo del robot

Esta idea ha surgido tras el estudio de diversos artículos de investigación como los que se muestran en el capítulo 10.

# 7.4 Integración de CATIA V5 con MATLAB

Tal y como se vio en el capítulo 5 sobre el funcionamiento de la interfaz, en un futuro proyecto fin de carrera se podría intentar unir las excelentes tareas que ofrece CATIA con la potente herramienta de cálculo que supone MATLAB.

CATIA ofrece una arquitectura de desarrollo de código abierto a través del uso de las interfaces, que se puede utilizar para personalizar o desarrollar aplicaciones. Los lenguajes de programación soportados son:

- Los lenguajes de programación Fortran y C para la versión 4 (V4).
- Visual Basic y C + + para la versión 5 (V5).

Estas API se conocen como CAA para V4 y CAA2 (o CAA V5) para V5. Los CAA2 son Component Object Model (COM) interfaces basadas en objetos. Estos proporcionan la integración de los productos desarrollados en la base del software CATIA.

Como MATLAB permite generar código y exportarlo como librerías de C++, sería interesante generar estas librerías de control cinemático y mediante un script en VBA o C++ usarlas en en el modelo CATIA del robot HOAP-3 que se ha creado en este proyecto.

# Capítulo 8

# Conclusiones y resultados

# 8.1 Resultados

En este apartado de resultados se va a presentar la respuesta de la interfaz en sus dos modalidades.

## 8.1.1 Resultados con el subprograma Manipulator Control

En primer lugar se ha generado dos trayectorias para el manipulador del brazo derecho y calculado su inversión cinemática.

**Trayectoria-1**: trayectoria o ruta circular. Desde la posición inicial con un incremento de la posición de 5 cm en dirección x, z, pasando por el punto vía [0,0,0] en 3 segundos. Véase Figura 8.1.1.

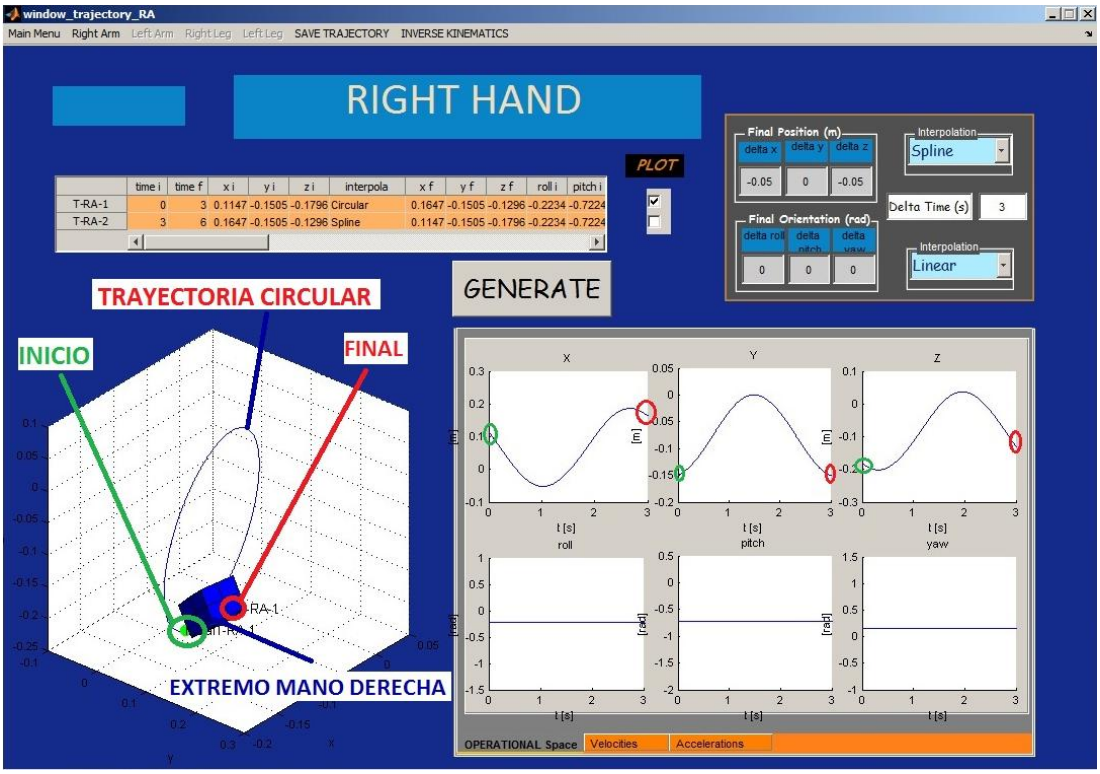



Figura 8.1.1: Trayectoria circular de la mano derecha

**Trayectoria-2**: trayectoria spline. Trayectoria spline desde la posición final de la trayectoria anterior con un incremento de la posición de -5 cm en dirección x,z.

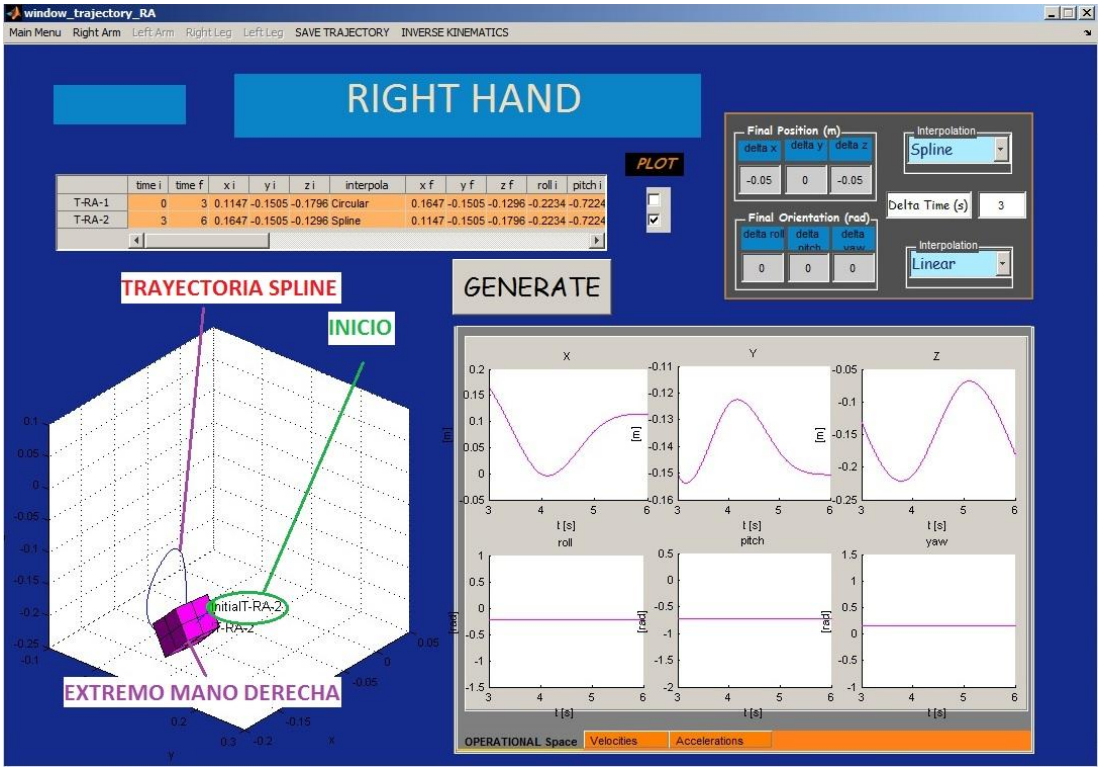



Figura 8.1.2: Trayectoria spline de la mano derecha

Se debe tener en cuenta que las velocidades y aceleraciones finales de la trayectoria circular, serán las velocidades iniciales en la trayectoria spline pero en el recuadro de recogida de datos de la trayectoria spline se tiene que poner estas velocidades y aceleraciones finales para imponer la continuidad en ambas. Véase siguiente figura, en ella se puede ver la continuidad de velocidades y aceleraciones.

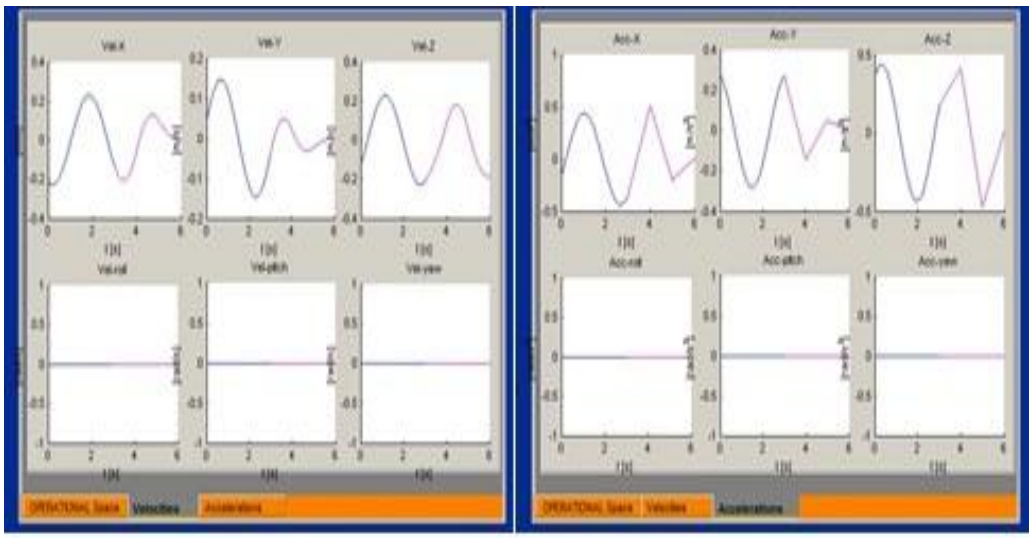


Figura 8.1.3: Velocidades y aceleraciones de Trayectoria 1 y 2

Al realizar la inversión cinemática y tras ver las gráficas del espacio articular del presente manipulador se observa en la Figura 8.1.4 que la trayectoria no se puede realizar ya que sobrepasa el intervalo máximo de valores de los motores del brazo derecho.

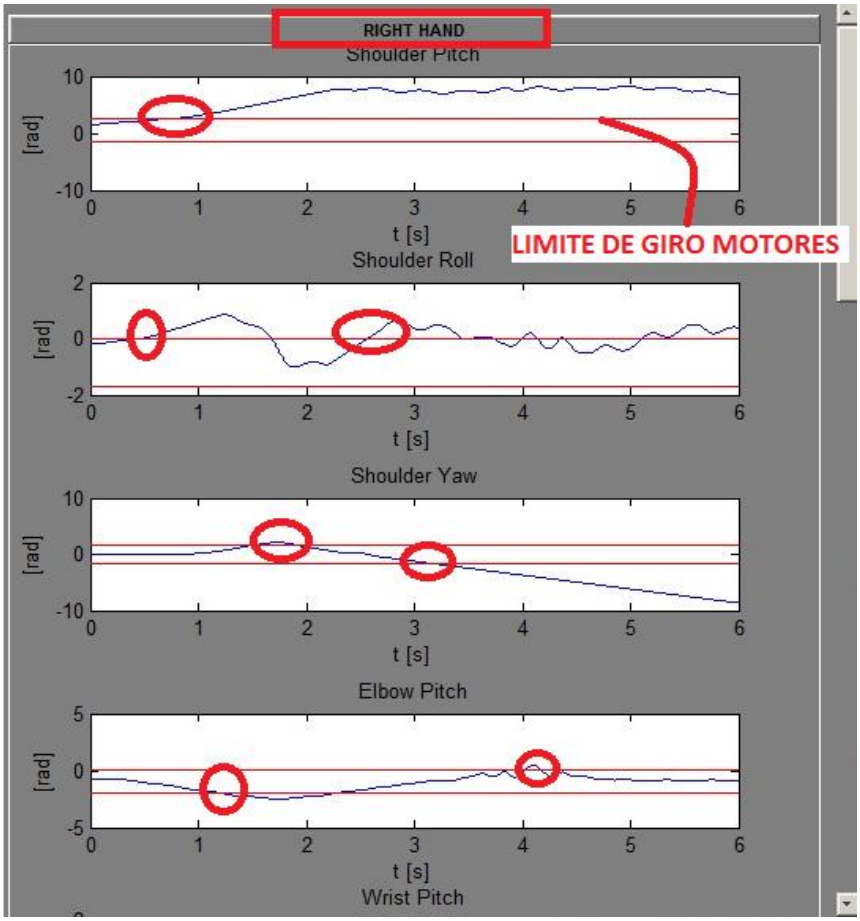



Figura 8.1.4: Ángulos de las articulaciones del brazo derecho

En la figura 8.1.4 se puede ver los límites de giro superior e inferior de los motores representados con dos líneas rojas paralelas al eje de tiempos. En color azul se representa la trayectoria articular de cada articulación. En este caso, salvo en el ángulo de giro de la muñeca (Wrist), en el resto de articulaciones la trayectoria articular sobrepasa los límites naturales impuestos.

Se decide por tanto por cambiar la interpolación de la Trayectoria 1 con un spline con el mismo incremento de posición e indicando una velocidad final en la dirección *y* de 0.1 m/s. La Trayectoria 2 se realiza con el mismo incremento pero negativo y forzando a que la velocidad inicial en la dirección *y* sea de 0.1 m/s, y las velocidades finales en las direcciones *x,z* también de 0.1 m/s. Véase Figura 8.1.5

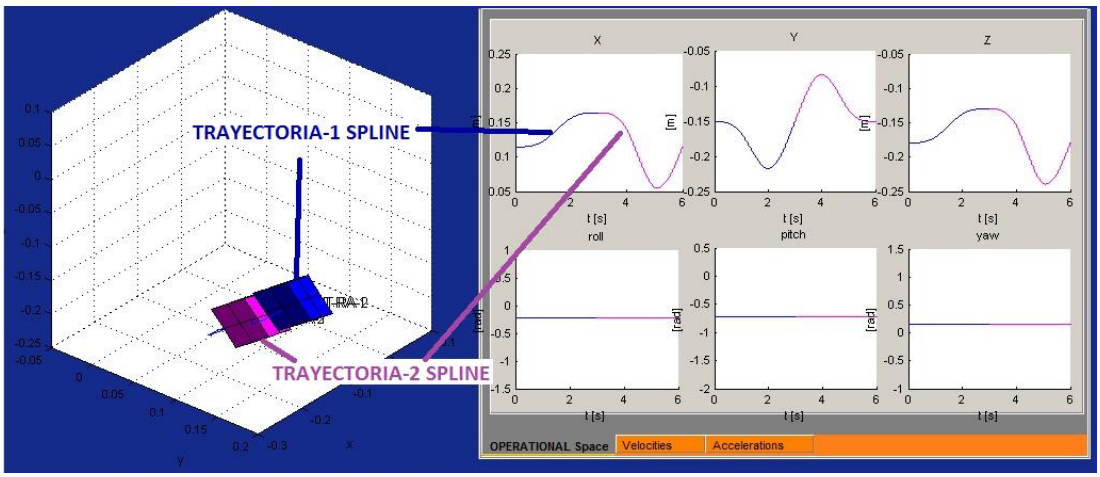



Figura 8.1.5: Trayectoria-1 spline y Trayectoria-2 spline

Al realizar la inversión cinemática se descubre que tampoco cumple con los criterios de límite de giro de los motores.

Finalmente se opta por realizar la Trayectoria 1 con una interpolación lineal a velocidad constante de 0.0167 m/s en las direcciones *x,y,z*. La Trayectoria 2 se mantiene con la interpolación spline e imponiendo la continuidad de velocidades como se hizo en las anteriores trayectorias. Véase Figura 8.1.6.

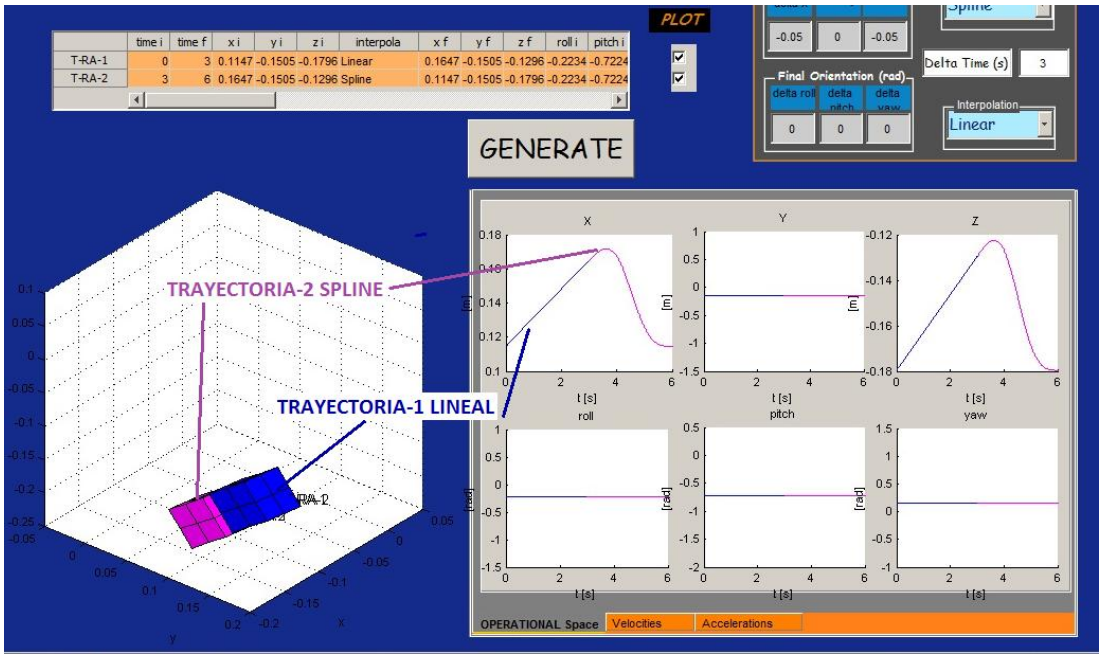



Figura 8.1.6: Trayectoria-1 lineal y Trayectoria-2 spline

Tras hacer la inversión cinemática, todos los ángulos del manipulador están dentro de los límites de los motores. Véase Figura 8.1.7 y Figura 8.1.8.

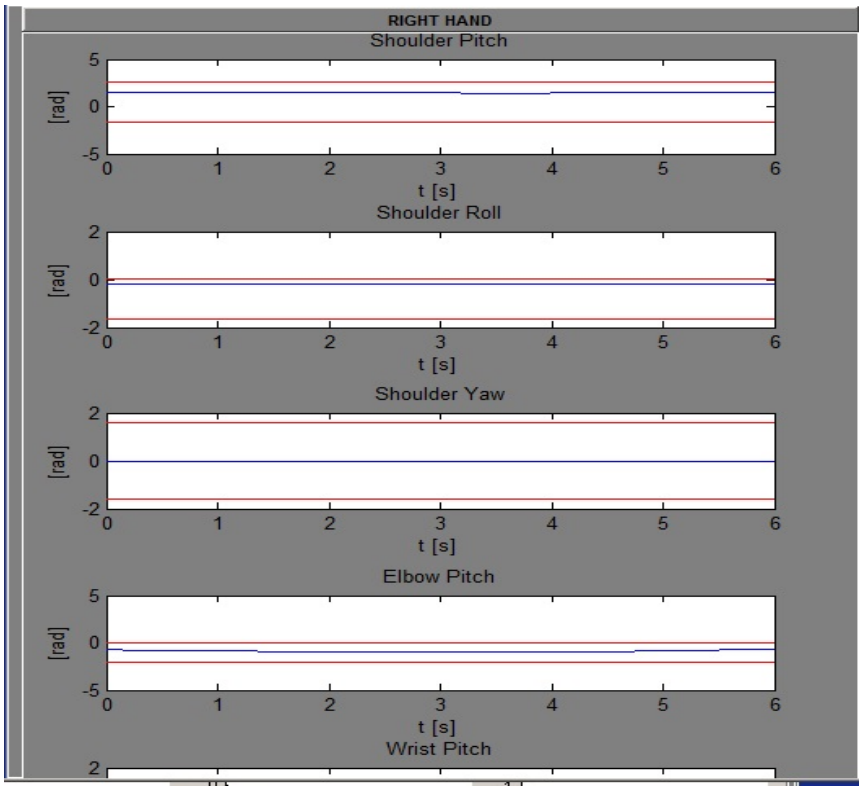



Figura 8.1.7: Trayectoria articular para el brazo derecho

En la figura de arriba no se percibe gran variación de los ángulos ya que no se ha escalado la representación de una forma adecuada. Si se observa la representación de las velocidades y aceleraciones (Figura 8.1.8) angulares se aprecia una variación sinusoidal que a continuación se discute.

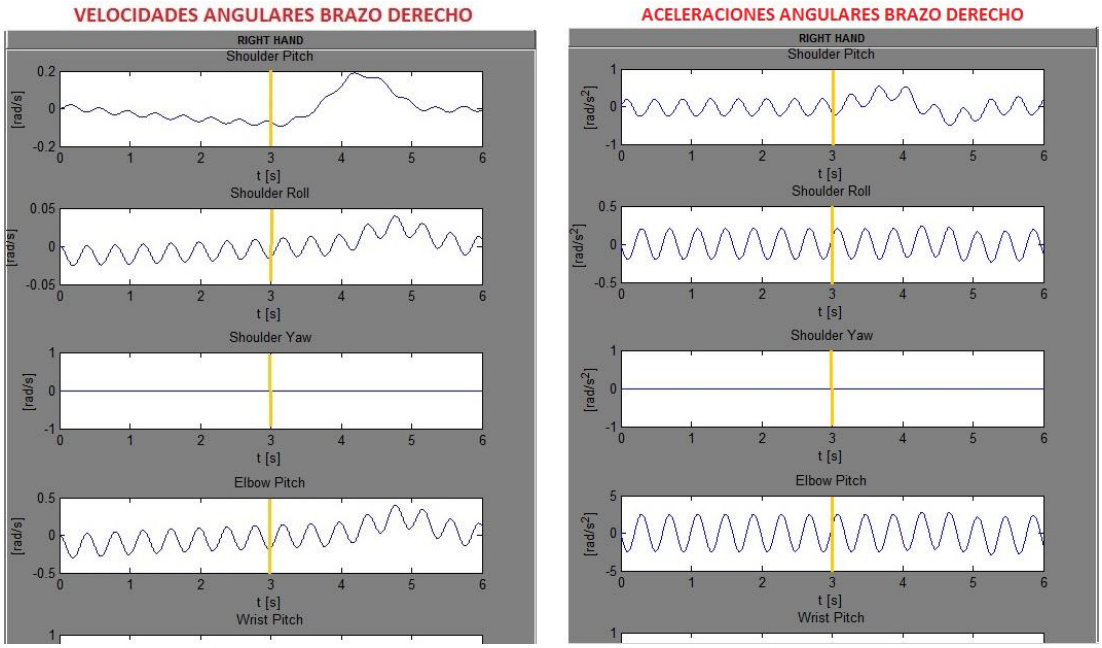



Figura 8.1.8: Velocidades y aceleraciones angulares

La forma que presentan la velocidad y aceleración del espacio articular indica que sí se produce movimiento en las articulaciones aunque no se aprecie bien en las gráficas de los ángulos de estas (figura 8.1.7). Según se vió en el capítulo 3 de análisis matemático a la hora de explicar la cinemática inversa y diferencial, en el algoritmo de resolución se está controlando un vínculo secundario a la vez que se controla la posición. Esto ocasiona una pequeña oscilación en la variable de control, que dependerá del peso que se le dé a este vínculo.

## 8.1.2 Resultados con el subprograma Step Control

A continuación se muestra un paso generado con Step Control y la simulación en Simulink. Será un paso de 10 cm de longitud y altura 5 cm, en 1.5 segundos. La mano derecha tendrá una interpolación lineal avanzando hacia delante y arriba de 5 cm, mientras que la mano izquierda irá hacia atrás y hacia abajo 2 cm.

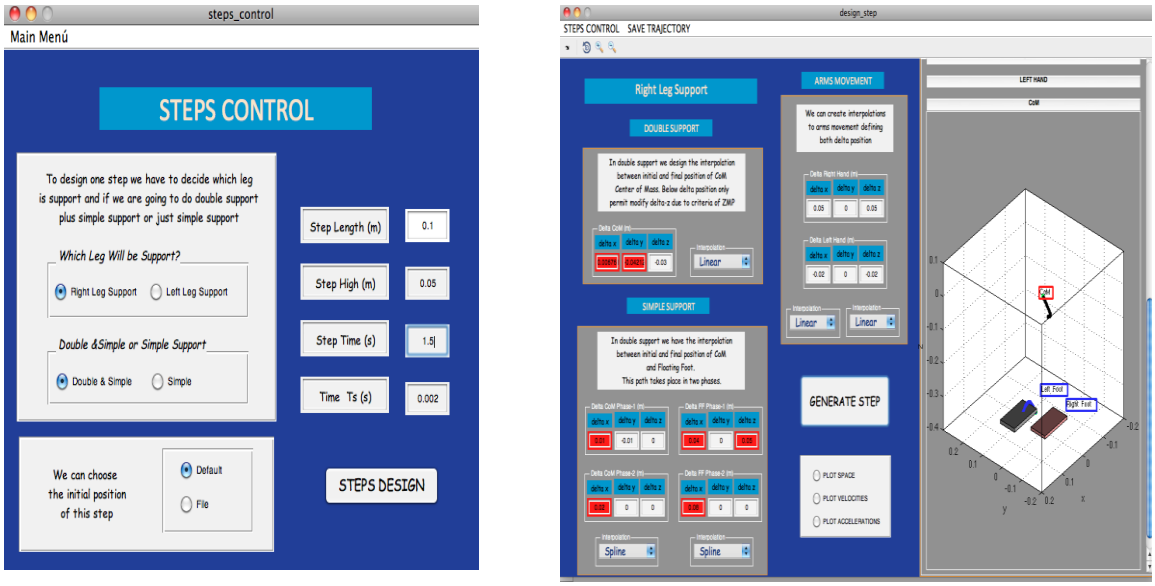



Figura 8.1.9: Paso generado con Step Control

En la Figura 8.1.9 se puede ver la simulación del movimiento de los pies y el centro de masas así como las gráficas de los ángulos del manipulador pierna izquierda y de sus velocidades angulares en la Figura 8.1.10.

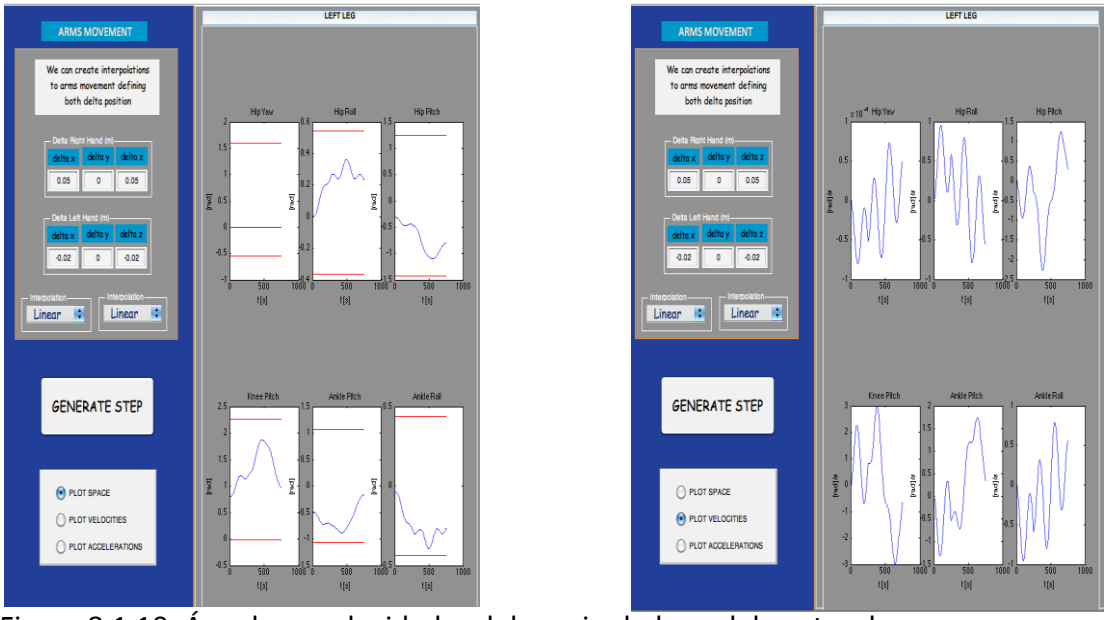



Figura 8.1.10: Ángulos y velocidades del manipulador y del centro de masas

En el modelo dinámico (realizado por Paolo Pierro en Simulink) que se ha empleado para contrastar los resultados, Figura 8.1.11, se puede observar cómo esta trayectoria generada es coherente con la interfaz y se encuentra dentro de los límites de giro de cada motor.

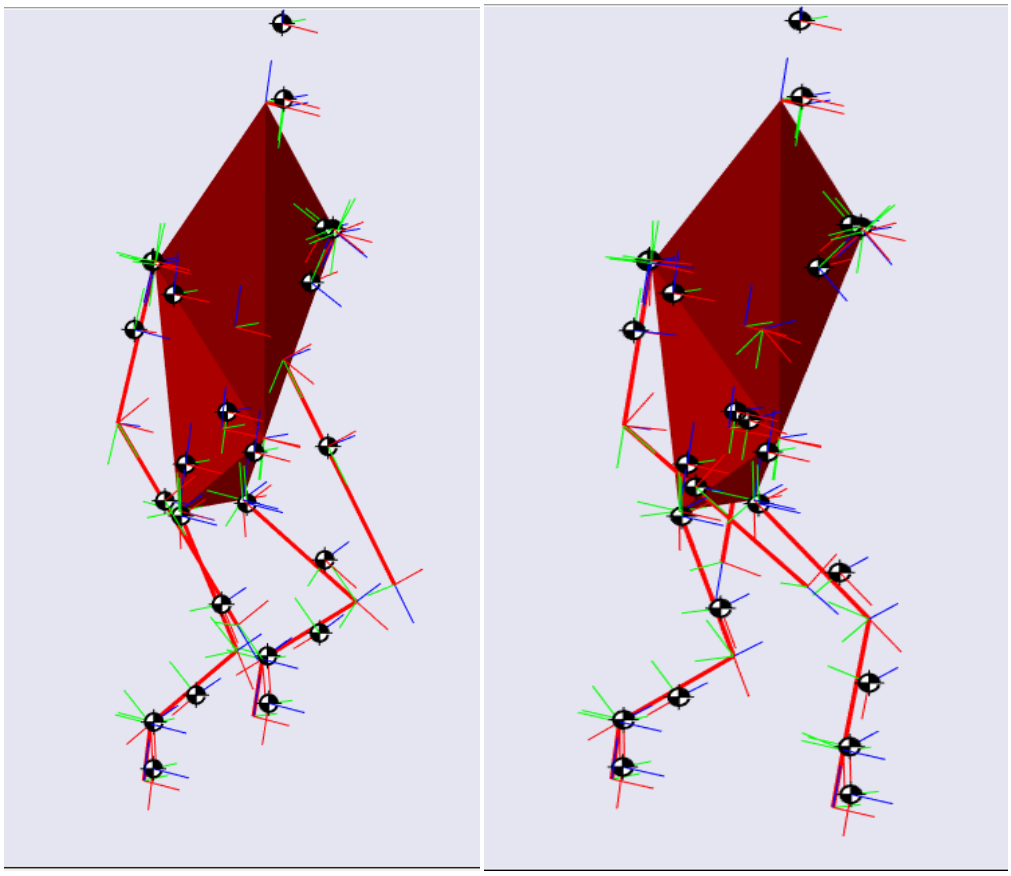


Figura 8.1.11: Simulación del paso en Simulink.

Como se indicó en el capítulo de trabajos futuros, este proyecto podría ser una buena base para realizar una futura interfaz de usuario de generación de caminatas para el robot humanoide Hoap-3. Tras haber realizado el paso anterior, se ha realizado una prueba continuando este paso tomando su posición final y su configuración articular como datos de partida para generar el siguiente paso.

Paso con doble y simple soporte desde el pie izquierdo, con una longitud de paso de 10 cm, altura del paso 5 cm, y tiempo del paso 3 segundos (véase siguiente figura, Figura 8.1.12).

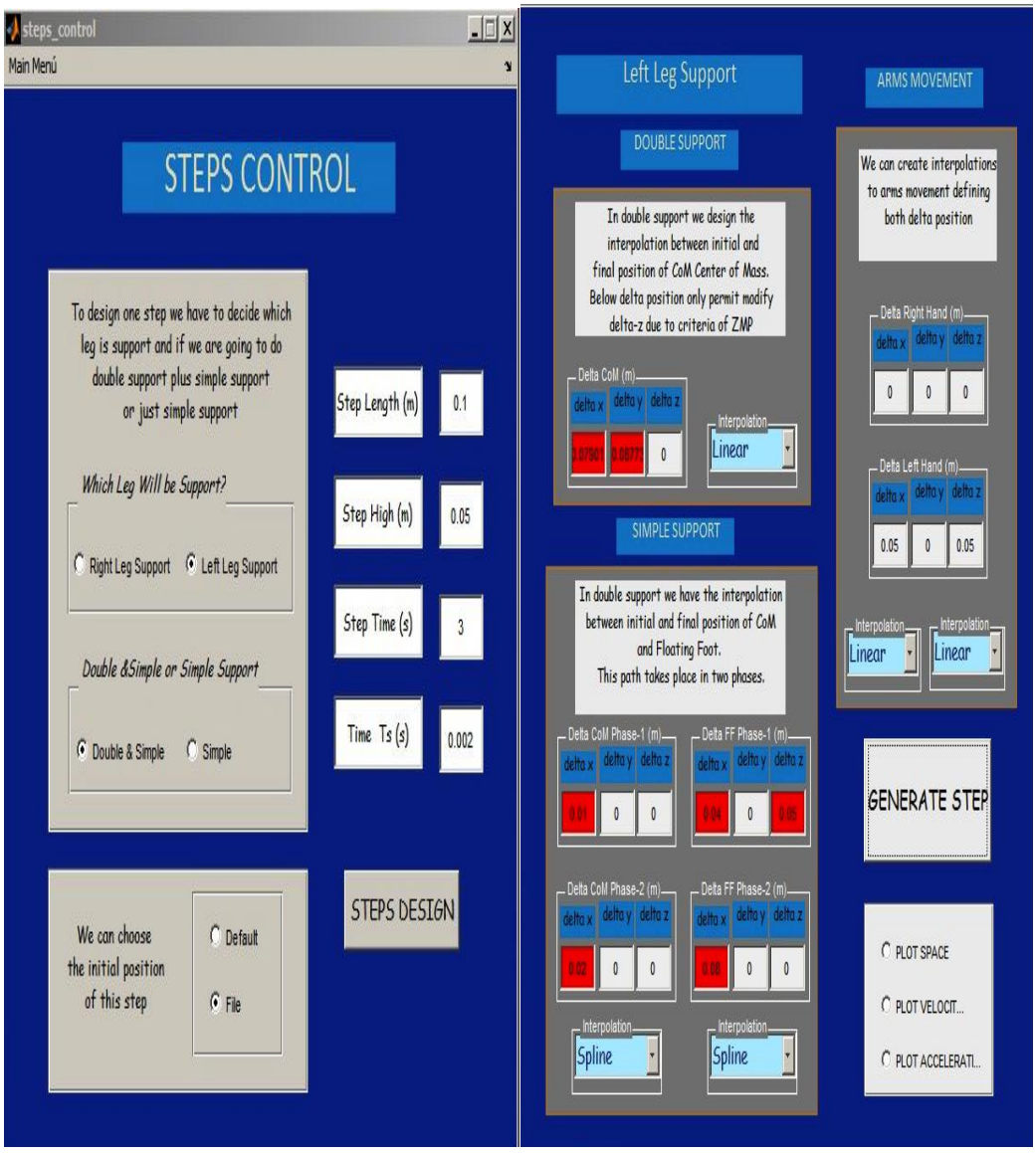


Figura 8.1.12 Segundo paso generado

Como se hizo en el primer paso, se puede ver la simulación del movimiento del centro de masas y de los pies en la siguiente figura. (Figura 8.1.13).

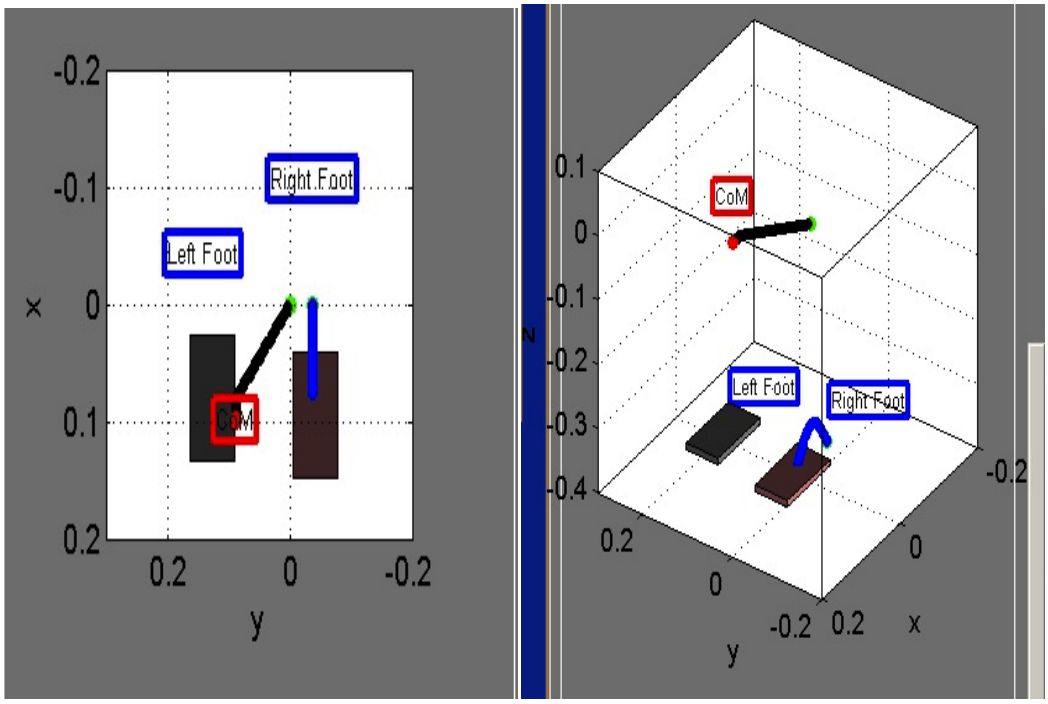



Figura 8.1.13 Simulación del movimiento de los pies y del centro de masas

Se observar que el pie izquierdo permanece como soporte y el pie derecho realiza su trayectoria de pie flotante. Intuitivamente, viendo el movimiento del centro de masas, se diría que este paso es estable al permanecer tal punto proyectado en plano x-y dentro de los límites de los pies.

En la posterior inversión cinemática, se descubre que el movimiento de la pierna derecha supera los límites de los ángulos de giro del tobillo derecho como se puede ver en la siguiente figura (Figura 8.1.14).

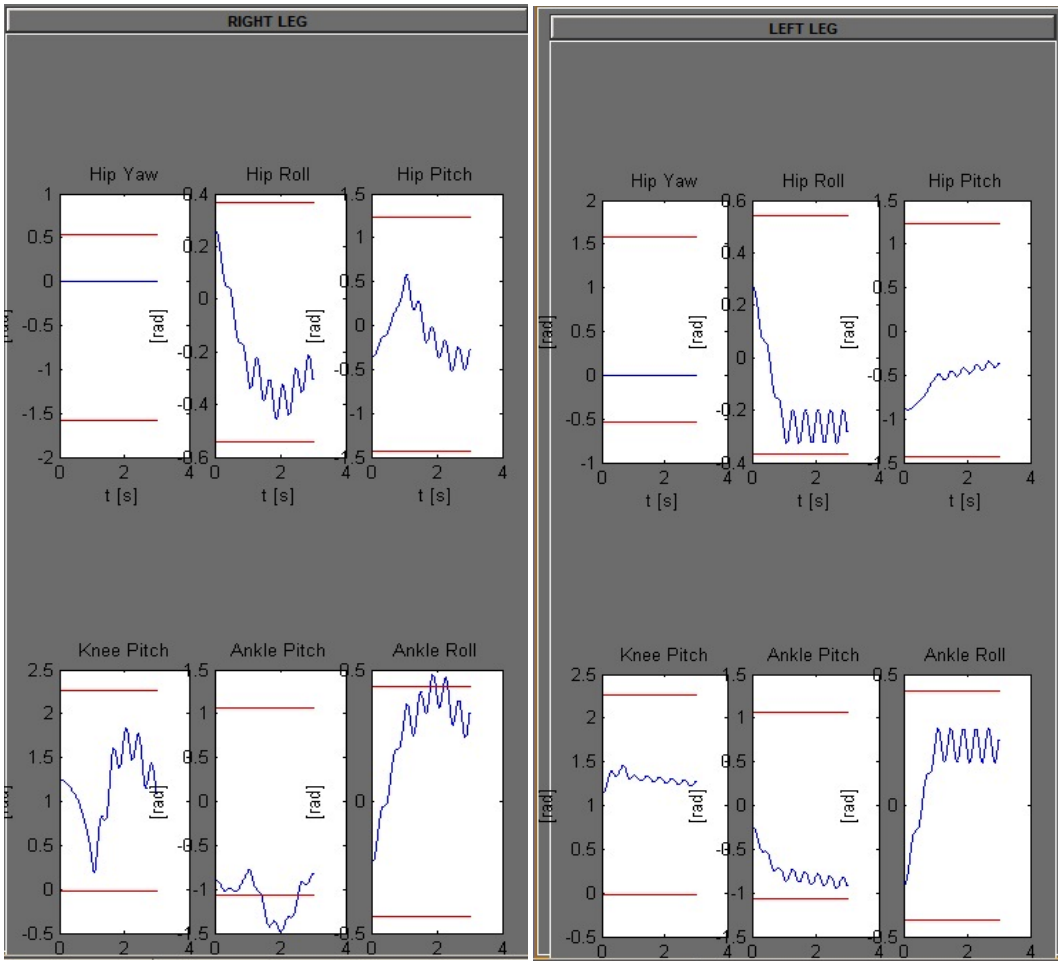



Figura 8.1.14 Espacio articular de la pierna derecha e izquierda

# 8.2 Conclusiones

La interfaz de generación de trayectorias ha demostrado ser una herramienta relativamente sencilla e intuitiva que permite trabajar con el robot de forma aislada controlando cada manipulador o bien de forma global generando un paso del mismo.

El poder trabajar con cada manipulador y estudiar diversas trayectorias, exportarlas y ver la simulación gráfica del extremo final del manipulador, puede ayudar en tareas futuras de diseño de movimientos específicos para el robot Hoap-3.

Si bien es cierto, al diseñar la interfaz se impusieron unas ciertas limitaciones hacia el usuario para no complicar demasiado su programación pero sobre todo para encaminar su uso claramente y guiada paso a paso.

Hay que procurar definir las trayectorias dentro de un rango posible de trabajo del robot, pues si no de otro modo la inversión cinemática no da resultados coherentes indicando que el usuario se encuentra fuera del rango posible de movimientos seguros.

Es conveniente observar las gráficas de las articulaciones para cerciorarse de que los movimientos exigidos no comprometen las posibilidades del robot. En estas gráficas se puede ver los límites de cada motor de tal modo que el usuario podrá saber si la trayectoria que desea que realice el robot es viable.

Llegar a conseguir que el robot encadene dos o más pasos consecutivos es una tarea muy complicada pero con esta herramienta que hoy se presenta es posible facilitar tal estudio y conseguir un resultado satisfactorio.

Destacar que las bases de esta interfaz se encuentran en la cinemática y por tanto los resultados que se obtengan de ella se deben de analizar con un posterior análisis dinámico para completar el estudio de movimientos del robot.

En el diseño del paso se ha tratado de incorporar limitaciones al movimiento del centro de masas, basadas en el ZMP, pero aún así esto no garantiza una completa estabilidad.

Las trayectorias que se generen deberán hacerse con pequeños incrementos de desplazamiento y orientación y con velocidades relativamente lentas, puesto que de lo contrario entrarían las leyes de la dinámica a formar parte de las trayectorias a realizar.

# Capítulo 9

# Bibliografía

# 9.1 Libros


- **[1]** pág 30 B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo : "Robotics. Modelling, Planning and Control". (Ed.Springer-Verlag). Año 2009.
- H. Goldstein, C. Poole, J. Safko: "Classical Mechanics". (Third Edition Addison Wesley) Año 2000.


# 9.2 Normativa


- Artículo 17 del BOE número 267 del jueves 5 de noviembre de 2009, sección 3 página 92526.


# 9.3 Apuntes de asignaturas


- "Métodos Numéricos". Departamento de Matemáticas. Universidad Carlos III de Madrid. Año 2008


# 9.4 Tesis y artículos de investigación


- Pardos Gotor, Jose Manuel.: 'Algoritmos de Geometría Diferencial para la Locomoción y Navegación Bípedas de Robots Humanoides. Aplicación al robot RH0'. Tesis Doctoral, Universidad Carlos III de Madrid, Departamento de Ingeniería de Sistemas y Automática, Año 2005.
- "Capturing Robot Workspace Structure: Representing Robot Capabilities", Franziska Zacharias, Christoph Borst y Gerd Hirzinger. Institute of Robotics and Mechatronics, German Aerospace Center (DLR),Germany, franziska.zacharias@dlr.de
- "Motion Planning Using Dynamic Roadmaps", Marcelo Kallmann and Maja Mataric' *Interaction Lab, Computer Science Department University of Southern California Los Angeles, California 90089-0781, USA* {kallmann, mataric}@usc.edu
- "Manipulation Planning with Workspace Goal Regions", The Robotics Institute, Carnegie Mellon University Intel Research Pittsburgh 5000 Forbes Ave., Pittsburgh, PA, 15213, USA.

# 9.5 Recursos electrónicos

## 9.5.1 Documentos electrónicos


- MATLAB: “Creating Graphical User Interfaces R2011b”. The MathWorks, Inc. Año 2011.


## 9.5.2 Páginas web


- Undocumented Matlab: http://undocumentedmatlab.com/
- Mathworks: http://www.mathworks.es
- Intech Open: http://www.intechopen.com
- Universidad de Nebrija: http://www.nebrija.es/~mjgarbayo/seminario_matlab/matlab12.html
- Ebook 3000: http://www.ebook3000.com/
- Biblioteca UC3M : http://biblioteca.uc3m.es/

# Anexo 1

# Código MATLAB

# Main Hoap

```
function varargout = main_hoap3(varargin)
% MAIN_HOAP3 MATLAB code for main_hoap3.fig
%      MAIN_HOAP3, by itself, creates a new MAIN_HOAP3 or raises the
existing
% Last Modified by GUIDE v2.5 05-Feb-2012 21:53:59
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_hoap3_OpeningFcn, ...
                   'gui_OutputFcn',  @main_hoap3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before main_hoap3 is made visible.
function main_hoap3_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;
%--------------------------------------------------------------------
-------
% To put screen correctly

scrsz = get(0,'ScreenSize');
pos_act = get(gcf,'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

% to chargue main image 'portada.jpg'
axes(handles.axes1)
[r,map]=imread('portada.jpg','jpg');
image(r);colormap(map);axis off
axes(handles.axes2)
[r,map]=imread('portada3.jpg','jpg');
image(r);colormap(map);axis off
% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
```

```
function varargout = main_hoap3_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

% --- Executes on button press in manipulator_control_button.
function manipulator_control_button_Callback(hObject, eventdata,
handles)

h = waitbar(0,'Please wait...');
steps = 300;

for step = 1:steps
    % computations take place here
    waitbar(step / steps)
end

settings_manipulator
close(h)
close main_hoap3

% --- Executes on button press in steps_control_button.
function steps_control_button_Callback(hObject, eventdata, handles)

steps_control

close main_hoap3
```

# Manipulator Control

settings_manipulator.m ----------

```
function varargout = settings_manipulator(varargin)
% **************************
% SETTINGS_MANIPULATOR MATLAB code for settings_manipulator.fig
% This window generates the main functions about manipulator control
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@settings_manipulator_OpeningFcn, ...
                   'gui_OutputFcn',
@settings_manipulator_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before settings_manipulator is made visible.
%                                                 ---------
settings_manipulator
function settings_manipulator_OpeningFcn(hObject, eventdata,
handles, varargin)
% -----------------
% Initial function
% -----------------
global SETTINGS_hoap

handles.output = hObject;
% ------------------------
scrsz = get(0,'ScreenSize');
pos_act = get(gcf,'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);
% ------------------------
axes(handles.axes1)
[r,map]=imread('tools2.jpg','jpg');
image(r);colormap(map);axis off
% ------------------------
```

```
SETTINGS_hoap.units.pos = '';
SETTINGS_hoap.units.orient = '';
SETTINGS_hoap.units.def = '';

SETTINGS_hoap.parameters.Ts = 0;
SETTINGS_hoap.parameters.kp = 0;
SETTINGS_hoap.parameters.kd = 0;

SETTINGS_hoap.save.fdat = 0;
SETTINGS_hoap.save.fcsv = 0;

SETTINGS_hoap.initial.from = '';
SETTINGS_hoap.initial.data = zeros(23,1);

SETTINGS_hoap.manipulators2use.RA = 0;
SETTINGS_hoap.manipulators2use.LA = 0;
SETTINGS_hoap.manipulators2use.RL = 0;
SETTINGS_hoap.manipulators2use.LL = 0;
% ------------------------
% Update handles structure
guidata(hObject, handles);


function varargout = settings_manipulator_OutputFcn(hObject,
eventdata, handles)

varargout{1} = handles.output;



% ************************** MENUS *****************************
% ------------------------------------------------------------------
--
function main_menu_of_settings_Callback(hObject, eventdata, handles)


main_hoap3
close settings_manipulator
%
*************************************************************************
*****
%                                                    ----
push_continue_Callback
function push_continue_Callback(hObject, eventdata, handles)

global SETTINGS_hoap manipulator_data trajectory d_trajectory
dd_trajectory Trajectory d_Trajectory dd_Trajectory
SETTINGS_hoap.parameters.Ts =
str2num(get(handles.Ts_edit,'String'));
SETTINGS_hoap.parameters.kp =
str2num(get(handles.kp_edit,'String'));
SETTINGS_hoap.parameters.kd =
str2num(get(handles.kd_edit,'String'));
CELDA_M = cell(4,2);
CELDA_M{2,1}= zeros(2,1); CELDA_M{2,2}=zeros(2,1);

CELDA_M{3,1}= {'' ''};CELDA_M{3,2}= {'' ''};
```

```
CELDA_M{4,1}= '';CELDA_M{4,2}= '';

manipulator_data.RA.datas = CELDA_M;
manipulator_data.RA.count_traj = zeros(1,2);
manipulator_data.LA.datas = CELDA_M;
manipulator_data.LA.count_traj = zeros(1,2);
manipulator_data.RLS.datas = CELDA_M;
manipulator_data.RLS.count_traj = zeros(1,2);
manipulator_data.RLF.datas = CELDA_M;
manipulator_data.RLF.count_traj = zeros(1,2);
manipulator_data.LLS.datas = CELDA_M;
manipulator_data.LLS.count_traj = zeros(1,2);
manipulator_data.LLF.datas = CELDA_M;
manipulator_data.LLF.count_traj = zeros(1,2);

% --------------------------------------------------------------------
-------
SETTINGS_hoap.h = hoap3_kinematics_library;
if exist('hoap3', 'var') == 0
    SETTINGS_hoap.hoap3 = hoap3_structure('numeric', 'rad', 'm');
end
SETTINGS_hoap.humanoid_fields = humanoid_operational_fields ();

trajectory = create_trajectory_template
(SETTINGS_hoap.humanoid_fields, (SETTINGS_hoap.parameters.Ts));
d_trajectory = create_trajectory_template
(SETTINGS_hoap.humanoid_fields, (SETTINGS_hoap.parameters.Ts));
dd_trajectory = create_trajectory_template
(SETTINGS_hoap.humanoid_fields, (SETTINGS_hoap.parameters.Ts));
q0 = SETTINGS_hoap.initial.data;
Ts = SETTINGS_hoap.parameters.Ts;
trajectory = insert_trajectory(trajectory,
SETTINGS_hoap.humanoid_fields,
create_trajectory_structure(pose_quat2rpy(SETTINGS_hoap.h.CoM_T_RH(q
0)), Ts, 0), 'RH');
trajectory = insert_trajectory(trajectory,
SETTINGS_hoap.humanoid_fields,
create_trajectory_structure(pose_quat2rpy(SETTINGS_hoap.h.CoM_T_LH(q
0)), Ts, 0), 'LH');
Trajectory = trajectory;
d_Trajectory = d_trajectory;
dd_Trajectory = dd_trajectory;
% --------------------------------------------------------------------
-------
if SETTINGS_hoap.manipulators2use.RA
    manipulator_data.RA.W_RA = window_trajectory_RA;
elseif SETTINGS_hoap.manipulators2use.LA
    manipulator_data.LA.W_LA = window_trajectory_LA;
elseif SETTINGS_hoap.manipulators2use.RL
    manipulator_data.RL.W_RL = window_trajectory_RL;
elseif SETTINGS_hoap.manipulators2use.LL
    manipulator_data.LL.W_LL = window_trajectory_LL;
end


guidata(hObject,handles)
close(handles.figure1)

% ------------------------- CALLBACKS ----------------------------
-------
```

```
% ----------- SELECTIONchangeFCN ------------------

% --- Executes when selected object is changed in position_unit.
function position_unit_SelectionChangeFcn(hObject, eventdata,
handles)

global SETTINGS_hoap
SETTINGS_hoap.units.pos = get(eventdata.NewValue,'String');

function orientation_unit_SelectionChangeFcn(hObject, eventdata,
handles)
global SETTINGS_hoap
SETTINGS_hoap.units.orient = get(eventdata.NewValue,'String');

function abs_diff_SelectionChangeFcn(hObject, eventdata, handles)
global SETTINGS_hoap
SETTINGS_hoap.units.def = get(eventdata.NewValue,'String');

function initial_data_panel_SelectionChangeFcn(hObject, eventdata,
handles)
global SETTINGS_hoap
ini_val = get(eventdata.NewValue,'String');
switch ini_val
    case 'default position'
        SETTINGS_hoap.initial.data = [0; 0.00325683448936741; -
0.308647699300050; 0.796421295515307; -0.487773596215257;
0.0278918646012491; 0; 0.00325683448936741; -0.311486990906165;
0.796421295515307; -0.484850796032492; -0.0354911450764397;
0.0349065850398866; 1.57079632679490; -0.167017153300893; 0; -
0.734875474523928; 0; 1.57079632679490; 0.167017153300893; 0; -
0.734875474523928; 0];
    case 'from file'
        [FileCSV PathCSV]=uigetfile({'*.csv'}, 'Choose initial csv
file');

        motors = csvread(FileCSV,size(load(FileCSV),1) - 1);

            data = (motors')*conversion;
        SETTINGS_hoap.initial.data =
[data(1:6);data(11:16);data(21);data(7:10);data(23);data(17:20);data
(23)];

end

% ------------ CHECKBOX -------------------

function check_filetxt_Callback(hObject, eventdata, handles)

global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.save.fdat =1;
else
    SETTINGS_hoap.save.fdat =0;
end

function check_filecsv_Callback(hObject, eventdata, handles)
global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.save.fcsv =1;
```

```
else
    SETTINGS_hoap.save.fcsv =0;
end

function check_LA_Callback(hObject, eventdata, handles)
global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.manipulators2use.LA =1;
else
    SETTINGS_hoap.manipulators2use.LA  =0;
end

function check_LL_Callback(hObject, eventdata, handles)
global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.manipulators2use.LL =1;
else
    SETTINGS_hoap.manipulators2use.LL =0;
end

function check_RA_Callback(hObject, eventdata, handles)
global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.manipulators2use.RA = 1;
else
    SETTINGS_hoap.manipulators2use.RA =0;

end

function check_RL_Callback(hObject, eventdata, handles)
global SETTINGS_hoap
if (get(hObject,'Value') == get(hObject,'Max'))
    SETTINGS_hoap.manipulators2use.RL =1;
else
    SETTINGS_hoap.manipulators2use.RL =0;
end

% ----------------------- Text & Edit ------------------------------
------

function Ts_edit_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Ts_edit_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function kp_edit_Callback(hObject, eventdata, handles)

function kp_edit_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function kd_edit_Callback(hObject, eventdata, handles)

function kd_edit_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

window_trajectory_RA.m ----------

```
function varargout = window_trajectory_RA(varargin)
% WINDOW_TRAJECTORY_RA MATLAB code for window_trajectory_RA.fig
%
%
% Author: Daniel J. GarcÌa-Cano Locatelli
% Initial Code for every window
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@window_trajectory_RA_OpeningFcn, ...
                   'gui_OutputFcn',
@window_trajectory_RA_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before window_trajectory_RA is made visible.
function window_trajectory_RA_OpeningFcn(hObject, eventdata,
handles, varargin)

global SETTINGS_hoap manipulator_data Trajectory d_Trajectory
dd_Trajectory

handles.output = hObject;
% ------------------------
scrsz = get(0,'ScreenSize');
pos_act = get(gcf,'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);
% ------------------------
graphstab = uitabpanel(...
  'Parent',handles.panel_graphics,...
  'Style','leftbottom',...
```

```
    'Units','normalized',...
    'Position',[0,0,1,1],...
    'FrameBackgroundColor',[1,0.5,0.1],...
    'FrameBorderType','beveledin',...
    'Title',{'OPERATIONAL Space','Velocities','Accelerations'},...
    'PanelHeights',[8,30,10,10],...
    'HorizontalAlignment','left',...
    'FontWeight','bold',...
    'TitleBackgroundColor',[1,0.5,0],...
    'TitleForegroundColor',[0,0,0],...
    'PanelBackgroundColor',[0.5,0.5,0.5],...
    'PanelBorderType','beveledout');

handles.hpanel1 = getappdata(graphstab,'panels');
set(handles.panel_s,'Parent',handles.hpanel1(1));
set(handles.panel_v,'Parent',handles.hpanel1(2));
set(handles.panel_a,'Parent',handles.hpanel1(3));
% ------------------------
handles.hunits = SETTINGS_hoap.units;
switch handles.hunits.pos
    case '[m] meters'
        set(handles.panel_position,'Title','Final Position
(m)','FontName','Comic');
    case '[mm] milimeters'
        set(handles.panel_position,'Title','Final Position
(mm)','FontName','Comic');
end
switch handles.hunits.orient
    case '[rad] radians'
        set(handles.panel_orientation,'Title','Final Orientation
(rad)','FontName','Comic');
    case '[deg] deg'
        set(handles.panel_orientation,'Title','Final Orientation
(deg)','FontName','Comic');
end
switch handles.hunits.def
    case 'Differential'
        set(handles.text_time,'String','Delta Time
(s)','FontName','Comic');
        set(handles.text_x,'String','delta x');
        set(handles.text_y,'String','delta y');
        set(handles.text_z,'String','delta z');
        set(handles.text_roll,'String','delta roll');
        set(handles.text_pitch,'String','delta pitch');
        set(handles.text_yaw,'String','delta yaw');
    case 'Absolute'
        set(handles.text_time,'String','Final Time
(s)','FontName','Comic');
        set(handles.text_x,'String','x f');
        set(handles.text_y,'String','y f');
        set(handles.text_z,'String','z f');
        set(handles.text_roll,'String','roll f');
        set(handles.text_pitch,'String','pitch f');
        set(handles.text_yaw,'String','yaw f');
end
% ------------------------
man = {'RA', 'LA', 'RL', 'LL'};
man_used = {};
for i=1:4
   if SETTINGS_hoap.manipulators2use.(man{i})
       [m,n] = size(man_used);
```

```
        man_used{n+1} = man{i};
    else
        set(handles.(strcat(man{i},'menu_traj')),'Enable','off');
    end
end

set(handles.table_traj,'Data',cell(1,40));
set(handles.text1,'String','MANIPULATOR CONTROL');
set(handles.RAmenu_traj,'Checked','on');
set([handles.panel_data handles.checkplot1
handles.checkplot2],'Visible','off');
set([handles.panel_s handles.panel_v
handles.panel_a],'Visible','off');
set(handles.axes_3d_RA,'Parent',handles.figure1,'Visible','off');

% Update handles structure

guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = window_trajectory_RA_OutputFcn(hObject,
eventdata, handles)
varargout{1} = handles.output;

% ************************** MENUS
************************************

% ------------------------------------------------------------------
--
function traj1_menu_Callback(hObject, eventdata, handles)
handles.man_act = 'RA';
handles.num_traj = 1;
trajec_function_initia(hObject,handles,handles.man_act,handles.num_t
raj)
set(handles.text1,'String','RIGHT HAND TRAJECTORY');
set(handles.panel_data,'Visible','on');
set(handles.traj2_menu,'Enable','on');
guidata(hObject,handles)

% ------------------------------------------------------------------
--
function traj2_menu_Callback(hObject, eventdata, handles)
handles.man_act = 'RA';
handles.num_traj = 2;
trajec_function_initia(hObject,handles,handles.man_act,handles.num_t
raj)
set(handles.text1,'String','RIGHT HAND TRAJECTORY');
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function LAmenu_traj_Callback(hObject, eventdata, handles)
global manipulator_data
manipulator_data.LA.W_LA = window_trajectory_LA;
guidata(hObject,handles)
close(handles.figure1)

% ------------------------------------------------------------------
```

```
--
function RLmenu_traj_Callback(hObject, eventdata, handles)
global manipulator_data
manipulator_data.RL.W_RL = window_trajectory_RL;
guidata(hObject,handles)
close(handles.figure1)

% --------------------------------------------------------------------
--
function LLmenu_traj_Callback(hObject, eventdata, handles)
global manipulator_data
manipulator_data.LL.W_LL = window_trajectory_LL;
guidata(hObject,handles)
close(handles.figure1)

% --------------------------------------------------------------------
--

% --------------------------------------------------------------------
--
function mmenu_traj_Callback(hObject, eventdata, handles)


settings_manipulator
guidata(hObject,handles)
close(handles.figure1)


% *************************** CALLBACKS
*********************************
%
% ------------- CHECKBOX PLOT ---------------
function checkplot1_Callback(hObject, eventdata, handles)
% hObject    handle to checkplot1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
plot_check(hObject,handles,1)
% Hint: get(hObject,'Value') returns toggle state of checkplot1

function checkplot2_Callback(hObject, eventdata, handles)
plot_check(hObject,handles,2)

function plot_check(hObject,handles,trajnum)
%
global SETTINGS_hoap manipulator_data Trajectory d_Trajectory
dd_Trajectory
% cla(handles.axes_portada);
% set(handles.axes_portada,'Visible','off');
set(handles.axes_3d_RA,'Visible','on');
panel_traj = {'s','v','a';'','d_','dd_'};
colnames = get(handles.table_traj, 'RowName');
colname = colnames{trajnum};


    if get(hObject, 'Value')

        colors = {'b','m'}; % Use consistent color for lines
```

```
        titulos = {'X' 'Y' 'Z' 'roll' 'pitch' 'yaw';...
        'Vel-X','Vel-Y','Vel-Z','Vel-roll','Vel-pitch','Vel-yaw';...
        'Acc-X','Acc-Y','Acc-Z','Acc-roll','Acc-pitch','Acc-yaw'};
        titulos_y = {'[m]' '[m]' '[m]' '[rad]' '[rad]' '[rad]';...
        '[m/s]' '[m/s]' '[m/s]' '[rad/s]' '[rad/s]' '[rad/s]';...
        '[m/s^2]' '[m/s^2]' '[m/s^2]' '[rad/s^2]' '[rad/s^2]'
'[rad/s^2]'};
        Ts = SETTINGS_hoap.parameters.Ts;

        %tt=
get_data_table(get(handles.table_traj,'Data'),'T',trajnum);
        tt = manipulator_data.RA.datas{2,trajnum};
        n_t0=round(tt(1)/Ts)+1;
        n_tf=round(tt(2)/Ts)+1;
       T.trajectory.data = Trajectory.RH;
       T.time = Trajectory.time(n_t0:n_tf);
       T.d_trajectory.data = d_Trajectory.RH;
       T.dd_trajectory.data = dd_Trajectory.RH;

        for jj=1:3

set(handles.(strcat('panel_',panel_traj{1,jj})),'Visible','on');

            for kk=1:6

set(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))),
'NextPlot', 'Add');


plot(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))),...
                        T.time,...

T.(strcat(panel_traj{2,jj},'trajectory')).data(kk,n_t0:n_tf),...
                    'DisplayName', colname,'Color',
colors{trajnum});hold on;

title(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))),titulos
(jj,kk))

xlabel(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))),'t
[s]')

ylabel(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))),titulo
s_y(jj,kk))
                    hold off;

            end

        end

        set(handles.axes_3d_RA,'NextPlot','Add');
        newplot(handles.axes_3d_RA);
        plot_movies(T.trajectory.data(:,n_t0:n_tf),trajnum,colname);

    else % Adding a line to the plot
        % Find the lineseries object and delete it
        for jj=1:3
            for kk=1:6
```

```
delete(findobj(handles.(strcat('axes_',panel_traj{1,jj},num2str(kk))
), 'DisplayName', colname))
            end
        end
        delete(findobj(handles.axes_3d_RA,'DisplayName',colname))
        %set(handles.axes_3d_RA_RA,'NewPlot','replacechildren');

    end

guidata(hObject,handles)

% ------------  POPUP INTERPOLA ----------------
function popup_interpola_orient_Callback(hObject, eventdata,
handles)


function popup_interpola_orient_CreateFcn(hObject, eventdata,
handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function popup_interpola_pos_Callback(hObject, eventdata, handles)


function popup_interpola_pos_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%
**********************************************************************
*****
% ------------- PUSH CALLBACK ----------------------

function push_generate_Callback(hObject, eventdata, handles)

global SETTINGS_hoap manipulator_data Trajectory d_Trajectory
dd_Trajectory p_via dp_ini dp_end ddp_ini ddp_end

Ts = SETTINGS_hoap.parameters.Ts;
p_via = zeros(3,1);dp_ini = zeros(3,1);dp_end = zeros(3,1);ddp_ini =
zeros(3,1);ddp_end = zeros(3,1);
contents = cellstr(get(handles.popup_interpola_pos,'String'));
handles.interp_pos =
contents{get(handles.popup_interpola_pos,'Value')};
contents = cellstr(get(handles.popup_interpola_orient,'String'));
handles.interp_orient =
contents{get(handles.popup_interpola_orient,'Value')};

data_act =
manipulator_data.(handles.man_act).datas{1,handles.num_traj};
T = manipulator_data.(handles.man_act).datas{2,handles.num_traj}; %
```

```
absolute time
manipulator_data.(handles.man_act).datas{3,handles.num_traj}={handle
s.interp_pos handles.interp_orient};

pf_edit =
[str2double(get(handles.x_edit,'String'));str2double(get(handles.y_e
dit,'String'));str2double(get(handles.z_edit,'String'))];
of_edit =
[str2double(get(handles.roll_edit,'String'));str2double(get(handles.
pitch_edit,'String'));str2double(get(handles.yaw_edit,'String'))];
tf_edit = str2double(get(handles.time_edit,'String'));

if strcmp(SETTINGS_hoap.units.def,'Differential')
    data_act(:,2) = data_act(:,1)+[pf_edit;of_edit];
    T(2) = T(1)+ tf_edit;
else
    data_act(:,2) = [pf_edit;of_edit];
    T(2) = tf_edit;
end

manipulator_data.(handles.man_act).datas{2,handles.num_traj}= T;
manipulator_data.(handles.man_act).datas{2,handles.num_traj+1}=
[T(2) T(2)];
% -------------------------- INTERPOLATION ------------------------
-------

% Position Interpolation

%data_act(:,:,2) =
transf_units(data_act(:,:,2),1,SETTINGS_hoap.units.pos,SETTINGS_hoap
.units.orient);


switch handles.interp_pos
    case 'Linear'

    case 'Circular'
       circular_points(data_act(1:3,1),data_act(1:3,2))
       uiwait

    case 'Spline'
       spline_points({'v0 x' 'v0 y' 'v0 z';'Vf x' 'Vf y' 'Vf
z'},{'a0 x' 'a0 y' 'a0 z';'Af x' 'Af y' 'Af z'});
       uiwait

       data_act(1:3,3) = dp_ini;
       data_act(1:3,4) = dp_end;
       data_act(1:3,5) = ddp_ini;
       data_act(1:3,6) = ddp_end;
    case 'Polynomial'
       polynomial_points({'v0 x' 'v0 y' 'v0 z';'Vf x' 'Vf y' 'Vf
z'});
       uiwait
       data_act(1:3,3) = dp_ini;
       data_act(1:3,4) = dp_end;

end
```

```
% Orientation Interpolation

dp_ini = zeros(3,1);dp_end = zeros(3,1);ddp_ini = zeros(3,1);ddp_end
= zeros(3,1);
switch handles.interp_orient
    case 'Linear'

    case 'Spline'
       spline_points({'v0 roll' 'v0 pitch' 'v0 yaw';'Vf roll' 'Vf
pitch' 'Vf yaw'},{'a0 row' 'a0 pitch' 'a0 yaw';'Af roll' 'Af pitch'
'Af yaw'});
       uiwait
       data_act(4:6,3) = dp_ini;
       data_act(4:6,4) = dp_end;
       data_act(4:6,5) = ddp_ini;
       data_act(4:6,6) = ddp_end;
    case 'Polynomial'
       polynomial_points({'v0 roll' 'v0 pitch' 'v0 yaw';'Vf roll'
'Vf pitch' 'Vf yaw'});
       uiwait
       data_act(4:6,3) = dp_ini;
       data_act(4:6,4) = dp_end;

end



data_table =
insert_data_table(get(handles.table_traj,'Data'),T,data_act(:,1),man
ipulator_data.(handles.man_act).datas{3,handles.num_traj},data_act(:
,2),data_act(:,3),data_act(:,4),data_act(:,5),data_act(:,6),handles.
num_traj);
set(handles.table_traj,'Data',data_table);

Delta_Interior =
transf_units(data_act,2,SETTINGS_hoap.units.pos,SETTINGS_hoap.units.
orient);
delta_P = Delta_Interior(:,2)-Delta_Interior(:,1);
d_delta_P = Delta_Interior(:,4)-Delta_Interior(:,3);
dd_delta_P = Delta_Interior(:,6)-Delta_Interior(:,5);

switch handles.man_act

    case {'RA'}

        [Trajectory, d_Trajectory, dd_Trajectory] =
moving_leg_arm('RH',{handles.interp_pos
handles.interp_orient},delta_P,d_delta_P,dd_delta_P,Ts, T,
Trajectory, d_Trajectory, dd_Trajectory);
    case {'LA'}
        [Trajectory, d_Trajectory, dd_Trajectory] =
moving_leg_arm('LH',{handles.interp_pos
handles.interp_orient},delta_P,d_delta_P,dd_delta_P,Ts, T,
Trajectory, d_Trajectory, dd_Trajectory);
```

```
    case {'RLS','LLS'}
        [Trajectory, d_Trajectory, dd_Trajectory] =
moving_leg_arm('CoM',{handles.interp_pos
handles.interp_orient},delta_P,d_delta_P,dd_delta_P,Ts, T,
Trajectory, d_Trajectory, dd_Trajectory);

    case 'RLF'
        [Trajectory, d_Trajectory, dd_Trajectory] =
moving_leg_arm('RF',{handles.interp_pos
handles.interp_orient},delta_P,d_delta_P,dd_delta_P,Ts, T,
Trajectory, d_Trajectory, dd_Trajectory);

    case 'LLF'
        [Trajectory, d_Trajectory, dd_Trajectory] =
moving_leg_arm('LF',{handles.interp_pos
handles.interp_orient},delta_P,d_delta_P,dd_delta_P,Ts, T,
Trajectory, d_Trajectory, dd_Trajectory);

end
% --------------------------------------------------------------------
-------

set(handles.(strcat('checkplot',num2str(handles.num_traj))),'Visible
','on');
set(handles.checkplot1,'TooltipString',strcat('Plot Trajectory','
',num2str(manipulator_data.RA.count_traj)));
guidata(hObject,handles)



%
**********************************************************************
*****
% -------------- INTERNAL FUNCTIONS ----------------

function trajec_function_initia(hObject,handles,manipulator,numtraj)
%
global SETTINGS_hoap manipulator_data Trajectory d_Trajectory
dd_Trajectory
q0 = SETTINGS_hoap.initial.data;
Ts = SETTINGS_hoap.parameters.Ts;


data_act = manipulator_data.(manipulator).datas{1,numtraj};
T = manipulator_data.(manipulator).datas{2,numtraj};
n = round(T(1)/Ts)+1;
data_act = zeros(6,6);
switch manipulator

    case 'RA'
        data_act(:,1) = Trajectory.RH(:,n);
        data_act(:,3) = d_Trajectory.RH(:,n);
        data_act(:,5) = dd_Trajectory.RH(:,n);
        set(handles.support_floating,'String','');
    case 'LA'
        data_act(:,1) = Trajectory.LH(:,n);
        data_act(:,3) = d_Trajectory.LH(:,n);
        data_act(:,5) = dd_Trajectory.LH(:,n);
        set(handles.support_floating,'String','');
```

```
    case 'RLS'
        if numtraj
           Trajectory = insert_trajectory(Trajectory,
SETTINGS_hoap.humanoid_fields,
create_trajectory_structure(pose_quat2rpy(SETTINGS_hoap.h.CoM_T_RF(q
0)), Ts, T(1)), 'RH');
           set([handles.traj1_menu_RLF
handles.traj1_menu_LLS],'Enable','off');
        else
           set([handles.traj2_menu_RLF
handles.traj2_menu_LLS],'Enable','off');
        end
        data_act(:,1) = Trajectory.CoM(:,n);
        data_act(:,3) = d_Trajectory.CoM(:,n);
        data_act(:,5) = dd_Trajectory.CoM(:,n);
        set(handles.support_floating,'String','Leg Support');
    case 'RLF'
        if numtraj
            data_act(:,1) =
pose_quat2rpy(SETTINGS_hoap.h.CoM_T_RF(q0));
            set([handles.traj1_menu_RLS],'Enable','off');
        else
            data_act(:,1) = Trajectory.RF(:,n);
            set([handles.traj2_menu_RLS],'Enable','off');
        end
        data_act(:,3) = d_Trajectory.RF(:,n);
        data_act(:,5) = dd_Trajectory.RF(:,n);
        set(handles.support_floating,'String','Leg Floating');

    case 'LLS'
        if numtraj
           Trajectory = insert_trajectory(Trajectory,
SETTINGS_hoap.humanoid_fields,
create_trajectory_structure(pose_quat2rpy(SETTINGS_hoap.h.CoM_T_RF(q
0)), Ts, T(1)), 'RH');
           set([handles.traj1_menu_LLF
handles.traj1_menu_RLS],'Enable','off');
        else
           set([handles.traj2_menu_LLF
handles.traj2_menu_RLS],'Enable','off');
        end

        data_act(:,1) = Trajectory.CoM(:,n);
        data_act(:,3) = d_Trajectory.CoM(:,n);
        data_act(:,5) = dd_Trajectory.CoM(:,n);
                set(handles.support_floating,'String','Leg
Support');
    case 'LLF'
        if numtraj
            data_act(:,1) =
pose_quat2rpy(SETTINGS_hoap.h.CoM_T_LF(q0));
            set([handles.traj1_menu_LLS],'Enable','off');
        else
            data_act(:,1) = Trajectory.LF(:,n);
            set([handles.traj2_menu_LLS],'Enable','off');
        end
        data_act(:,3) = d_Trajectory.LF(:,n);
        data_act(:,5) = dd_Trajectory.LF(:,n);
        set(handles.support_floating,'String','Leg Floating');
end
data_act =
```

```
transf_units(data_act,1,SETTINGS_hoap.units.pos,SETTINGS_hoap.units.
orient);
manipulator_data.(manipulator).datas{1,numtraj} = data_act;
manipulator_data.(manipulator).datas{4,numtraj} = strcat('T-
',manipulator,'-',num2str(numtraj));
data_table =
insert_data_table(get(handles.table_traj,'Data'),T,data_act(:,1),{''
''},data_act(:,2),data_act(:,3),data_act(:,4),data_act(:,5),data_act
(:,6),numtraj);
set(handles.table_traj,'Data',data_table,'RowName',manipulator_data.
(manipulator).datas(4,1:numtraj));
guidata(hObject,handles)

function data2 = transf_units(data1,caso,caso_pos,caso_orient)
%
% caso_pos and caso_orient are always table state
MC_m2mm = 1000*eye(3);
MC_mm2m = 0.001*eye(3);
MC_rad2deg = (180/pi)*eye(3);
MC_deg2rad = (pi/180)*eye(3);
if caso % to transform from internal to table (caso 1)
    switch caso_pos
        case '[m] meters'
            Mp = eye(3);

        case '[mm] milimeters'
            Mp = MC_m2mm;
    end

    switch caso_orient
        case '[rad] radians'
            Mo = eye(3);

        case '[deg] deg'
            Mo = MC_rad2deg;
    end
else % to transform from table to internal data(caso 0)
    switch caso_pos
        case '[m] meters'
            Mp = eye(3);

        case '[mm] milimeters'
            Mp = MC_mm2m;
    end

    switch caso_orient
        case '[rad] radians'
            Mo = eye(3);

        case '[deg] deg'
            Mo = MC_def2rad;
    end


end
data2 = [Mp zeros(3);zeros(3) Mo]*data1;


% --------------------------------------------------------------------
```

```
--
function IKmenu_traj_Callback(hObject, eventdata, handles)
% hObject    handle to IKmenu_traj (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
window_inverse_kinematics
guidata(hObject,handles)
close(handles.figure1)
```

# Step Control

steps_control.m ----------

```
function varargout = steps_control(varargin)
% STEPS_CONTROL MATLAB code for steps_control.fig

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @steps_control_OpeningFcn, ...
                   'gui_OutputFcn',  @steps_control_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before steps_control is made visible.
function steps_control_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;

% ------------------------------------------------------------------
-------
scrsz = get(0,'ScreenSize');
pos_act = get(gcf,'Position');
xr = scrsz(3) - pos_act(3);
xp = round(xr/2);
yr = scrsz(4) - pos_act(4);
yp = round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);
% ------------------------------------------------------------------
-------

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = steps_control_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;


% --- Executes on button press in step_button.
function step_button_Callback(hObject, eventdata, handles)
```

```
conversion = pi/180/209;
Input_data.Leg = handles.leg_support;
Input_data.DS_or_SS = handles.ds_or_ss;
Input_data.L_val = str2double(get(handles.L,'String'));
Input_data.H_val = str2double(get(handles.H,'String'));
Input_data.T_val = str2double(get(handles.T,'String'));
Input_data.Ts_val = str2double(get(handles.Ts,'String'));
Input_data.alpha_ds = 1/3;
Input_data.alpha_sf = 0.2;
switch handles.data_choose

    case 'Default'

        Input_data.q0 = [0; 0.00325683448936741; -
0.308647699300050;...
            0.796421295515307; -0.487773596215257;
0.0278918646012491;...
            0; 0.00325683448936741; -0.311486990906165;...
            0.796421295515307;...
            -0.484850796032492; -0.0354911450764397;
0.0349065850398866;...
            1.57079632679490; -0.167017153300893; 0; -
0.734875474523928;...
            0; 1.57079632679490; 0.167017153300893; 0;...
            -0.734875474523928; 0];

    case 'File' % This read the last row

        [FileCSV PathCSV]=uigetfile({'*.csv'}, 'Choose initial csv
file');

        motors = csvread(FileCSV,size(load(FileCSV),1) - 1);


        data = (motors')*conversion;
        Input_data.q0 =
[data(1:6);data(11:16);data(21);data(7:10);...
            data(23);data(17:20);data(23)];

end

design_step(Input_data)

close(handles.figure1)




% ------------------------------------------------------------------
--
function main_menu_Callback(hObject, eventdata, handles)

main_hoap3
close(handles.figure1)
```

```
% --- Executes when selected object is changed in leg_support.
function leg_support_SelectionChangeFcn(hObject, eventdata, handles)
new_val = get(eventdata.NewValue,'String');
old_val = get(eventdata.OldValue,'String');
handles.leg_support = new_val;
guidata(hObject,handles)


% --- Executes when selected object is changed in choose_data.
function choose_data_SelectionChangeFcn(hObject, eventdata, handles)

new_val = get(eventdata.NewValue,'String');
old_val = get(eventdata.OldValue,'String');
handles.data_choose = new_val;
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function L_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function H_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
```

```
function T_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function Ts_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes when selected object is changed in double_or_simple.
function double_or_simple_SelectionChangeFcn(hObject, eventdata,
handles)
new_val = get(eventdata.NewValue,'String');
old_val = get(eventdata.OldValue,'String');
handles.ds_or_ss = new_val;
guidata(hObject,handles)
```

---

**design_step.m**

```
function varargout = design_step(varargin)
% DESIGN_STEP MATLAB code for design_step.fig
%
%
% Last Modified by GUIDE v2.5 19-Feb-2012 13:36:47
%
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @design_step_OpeningFcn, ...
                   'gui_OutputFcn',  @design_step_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before design_step is made visible.
function design_step_OpeningFcn(hObject, eventdata, handles,
varargin)
% Choose default command line output for design_step
global h hoap3 pvia
handles.output = hObject;
handles.Input_data = varargin{:};
L = handles.Input_data.L_val;
H = handles.Input_data.H_val;
q0 = handles.Input_data.q0;
Leg = handles.Input_data.Leg;
set(handles.text1_design,'String',Leg);
DS_or_SS = handles.Input_data.DS_or_SS;
alpha_sf = handles.Input_data.alpha_sf;
% Load Hoap3 Kinematics Libray ----------------------------------
LIBRARY

h = hoap3_kinematics_library;

hoap3 = hoap3_structure('numeric', 'rad', 'm');
if strcmp(DS_or_SS,'Simple')
    set(handles.double_support_data_panel,'Visible','off');
    set(handles.text2_design,'String','No Double Support');
else
    set(handles.double_support_data_panel,'Visible','on');
end

% ------------------------------------
% to pre-evaluate delta x and delta y CoM in Double Support
switch Leg
    case 'Right Leg Support' % Support on right foot
        delta = h.CoM_T_RF(q0);
        delta_P(1,:) = (2*delta(1) +
(hoap3.legs.right.foot.limits.x(1) +
hoap3.legs.right.foot.limits.x(2))/1);
        delta_P(2,:) = (2*delta(2) +
(hoap3.legs.right.foot.limits.y(1) +
hoap3.legs.right.foot.limits.y(2))/1);

    case 'Left Leg Support' % Support on left foot
        delta = h.CoM_T_LF(q0);
        delta_P(1,:) = (2*delta(1) +
(hoap3.legs.left.foot.limits.x(1) +
hoap3.legs.left.foot.limits.x(2))/1);
        delta_P(2,:) = (2*delta(2) +
(hoap3.legs.left.foot.limits.y(1) +
hoap3.legs.left.foot.limits.y(2))/1);

end
delta_P = delta_P/2;
set(handles.delta_x_CoM_DS,'String',num2str(delta_P(1)),'BackgroundC
olor','red');
set(handles.delta_y_CoM_DS,'String',num2str(delta_P(2)),'BackgroundC
olor','red');

X1_SF = L*alpha_sf/2;
set(handles.delta_x_CoM_SS1,'String',num2str(X1_SF),'BackgroundColor
```

```
','red');
X2_SF = L*alpha_sf;
set(handles.delta_x_CoM_SS2,'String',num2str(X2_SF),'BackgroundColor
','red');

X1_FF = L*(1-alpha_sf)/2;
set(handles.delta_x_FF_SS1,'String',num2str(X1_FF),'BackgroundColor'
,'red');
set(handles.delta_z_FF_SS1,'String',num2str(H),'BackgroundColor','re
d');

X2_FF = L*(1-alpha_sf);
set(handles.delta_x_FF_SS2,'String',num2str(X2_FF),'BackgroundColor'
,'red');

pvia = zeros(3,1);

graphstab = uitabpanel(...
  'Parent',handles.panel_graphics,...
  'Style','popup',...
  'Units','normalized',...
  'Position',[0,0,1,1],...
  'FrameBackgroundColor',[0.5,0.5,0.5],...
  'FrameBorderType','etchedin',...
  'Title',{'RIGHT LEG','LEFT LEG','RIGHT HAND','LEFT
HAND','CoM'},...
  'PanelHeights',[59.5,59.5,50,50,50],...
  'HorizontalAlignment','left',...
  'FontWeight','bold',...
  'TitleBackgroundColor',[0.5,0.5,0.5],...
  'TitleForegroundColor',[0 0 0],...
  'PanelBackgroundColor',[0.5,0.5,0.5],...
  'PanelBorderType','line','SelectedItem',5);

hpanel = getappdata(graphstab,'panels');

% ******************************

   handles.sd_graph =
axes('Parent',getappdata(graphstab,'status'),...
    'Units','normalized',...
      'Position',[0.2,0.2,0.6,0.6],'Box','on');
   handles.axes1 = axes('Parent',hpanel(1),'Position',[.1 .6 .25
.25]);
   handles.axes2 = axes('Parent',hpanel(1),'Position',[.4 .6 .25
.25]);
   handles.axes3 = axes('Parent',hpanel(1),'Position',[.7 .6 .25
.25]);
   handles.axes4 = axes('Parent',hpanel(1),'Position',[.1 .15 .25
.25]);
   handles.axes5 = axes('Parent',hpanel(1),'Position',[.4 .15 .25
.25]);
   handles.axes6 = axes('Parent',hpanel(1),'Position',[.7 .15 .25
.25]);

   handles.axes7 = axes('Parent',hpanel(2),'Position',[.1 .6 .25
.25]);
   handles.axes8 = axes('Parent',hpanel(2),'Position',[.4 .6 .25
.25]);
```

```
   handles.axes9 = axes('Parent',hpanel(2),'Position',[.7 .6 .25
.25]);
   handles.axes10 = axes('Parent',hpanel(2),'Position',[.1 .15 .25
.25]);
   handles.axes11 = axes('Parent',hpanel(2),'Position',[.4 .15 .25
.25]);
   handles.axes12 = axes('Parent',hpanel(2),'Position',[.7 .15 .25
.25]);

   handles.axes14 = axes('Parent',hpanel(3),'Position',[.1 .6 .25
.25]);
   handles.axes15 = axes('Parent',hpanel(3),'Position',[.4 .6 .25
.25]);
   handles.axes16 = axes('Parent',hpanel(3),'Position',[.7 .6 .25
.25]);
   handles.axes17 = axes('Parent',hpanel(3),'Position',[.1 .15 .25
.25]);
   handles.axes18 = axes('Parent',hpanel(3),'Position',[.4 .15 .25
.25]);


   handles.axes19 = axes('Parent',hpanel(4),'Position',[.1 .6 .25
.25]);
   handles.axes20 = axes('Parent',hpanel(4),'Position',[.4 .6 .25
.25]);
   handles.axes21 = axes('Parent',hpanel(4),'Position',[.7 .6 .25
.25]);
   handles.axes22 = axes('Parent',hpanel(4),'Position',[.1 .15 .25
.25]);
   handles.axes23 = axes('Parent',hpanel(4),'Position',[.4 .15 .25
.25]);

   handles.axes_zmp = axes('Parent',hpanel(5),'Position',[.2 .2 .6
.6]);
   axes(handles.sd_graph)
   [r,map]=imread('portada.jpg','jpg');
   image(r);colormap(map);axis off
   axes(handles.axes_zmp)
   [r,map]=imread('portada.jpg','jpg');
   image(r);colormap(map);axis off
% ******************************
% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = design_step_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

% Callbacks ---------------------------------------------------------
-------
% --- Executes on selection change in interpolaSScom.
function interpolaSScom_Callback(hObject, eventdata, handles)
contents = cellstr(get(hObject,'String'));
handles.interpola_SS_com = contents{get(hObject,'Value')};
guidata(hObject,handles)

% --- Executes on selection change in interpolaDS.
```

```
function interpolaDS_Callback(hObject, eventdata, handles)
contents = cellstr(get(hObject,'String'));
handles.interpola_DS = contents{get(hObject,'Value')};

% --- Executes on selection change in interpolaSSff.
function interpolaSSff_Callback(hObject, eventdata, handles)
contents = cellstr(get(hObject,'String'));
handles.interpola_SS_ff = contents{get(hObject,'Value')};
guidata(hObject,handles)

% --- Executes on selection change in interpolaLH.
function interpolaLH_Callback(hObject, eventdata, handles)

contents = cellstr(get(hObject,'String'));
handles.interpola_LH = contents{get(hObject,'Value')};
guidata(hObject,handles)

% --- Executes on selection change in interpolaRH.
function interpolaRH_Callback(hObject, eventdata, handles)
contents = cellstr(get(hObject,'String'));
handles.interpola_RH = contents{get(hObject,'Value')};

guidata(hObject,handles)

% --- Executes on button press in generate_button.
function generate_button_Callback(hObject, eventdata, handles)
% hObject    handle to generate_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Steps Data
data.TS = handles.Input_data.Ts_val;
data.t0 = 0;
data.T = handles.Input_data.T_val;
%T = 6;
data.alpha_ds = handles.Input_data.alpha_ds;
data.alpha_sf = handles.Input_data.alpha_sf;
data.DS_or_SS = handles.Input_data.DS_or_SS;
data.L = handles.Input_data.L_val;
data.H = handles.Input_data.H_val;
data.q0 = handles.Input_data.q0;

% Delta Data

delta.delta_CoM_DS =
[str2double(get(handles.delta_x_CoM_DS,'String'));...
    str2double(get(handles.delta_y_CoM_DS,'String'));...
    str2double(get(handles.delta_z_CoM_DS,'String'));zeros(3,1)];

delta.interpola_CoM_DS = handles.interpola_DS;


delta.delta_CoM_SS1 =
[str2double(get(handles.delta_x_CoM_SS1,'String'));...
    str2double(get(handles.delta_y_CoM_SS1,'String'));...
    str2double(get(handles.delta_z_CoM_SS1,'String'));zeros(3,1)];

delta.delta_CoM_SS2 =
```

```
[str2double(get(handles.delta_x_CoM_SS2,'String'));...
    str2double(get(handles.delta_y_CoM_SS2,'String'));...
    str2double(get(handles.delta_z_CoM_SS2,'String'));zeros(3,1)];

delta.interpola_CoM_SS = handles.interpola_SS_com;


delta.delta_FF_SS1 =
[str2double(get(handles.delta_x_FF_SS1,'String'));...
    str2double(get(handles.delta_y_FF_SS1,'String'));...
    str2double(get(handles.delta_z_FF_SS1,'String'));zeros(3,1)];

delta.delta_FF_SS2 =
[str2double(get(handles.delta_x_FF_SS2,'String'));...
    str2double(get(handles.delta_y_FF_SS2,'String'));...
    str2double(get(handles.delta_z_FF_SS2,'String'));zeros(3,1)];

delta.interpola_FF_SS = handles.interpola_SS_ff;


delta.delta_RH = [str2double(get(handles.delta_x_RH,'String'));...
    str2double(get(handles.delta_y_RH,'String'));...
    str2double(get(handles.delta_z_RH,'String'));zeros(3,1)];

delta.interpola_RH = handles.interpola_RH;

delta.delta_LH = [str2double(get(handles.delta_x_LH,'String'));...
    str2double(get(handles.delta_y_LH,'String'));...
    str2double(get(handles.delta_z_LH,'String'));zeros(3,1)];

delta.interpola_LH = handles.interpola_LH;

 barra= waitbar(0,'Please wait...');
steps = 300;



for step = 1:steps
    % computations take place here
    waitbar(step / steps)
end
[q,dq,ddq,trajectory,d_trajectory,dd_trajectory] =
ds_ss_step_hoap(delta,data,handles.Input_data.Leg);
close(barra)



cla(handles.axes_zmp)

handles.result.q = q;
handles.result.dq = dq;
handles.result.ddq = ddq;
handles.result.trajectory = trajectory;
handles.result.d_trajectory = d_trajectory;
handles.result.dd_trajectory = dd_trajectory;

set(handles.axes_zmp,'NextPlot','Add');
```

```
axis on
newplot(handles.axes_zmp);

prueba_ZMP(handles)



guidata(hObject,handles)


% --- Executes when selected object is changed in plot_options.
function plot_options_SelectionChangeFcn(hObject, eventdata,
handles)

handles.plot_new = get(eventdata.NewValue,'String');
old_val = get(eventdata.OldValue,'String');

for jj =1:6
            cla(handles.(strcat('axes',num2str(jj))))
            cla(handles.(strcat('axes',num2str(jj+6))))


end
for jj=1:5
            cla(handles.(strcat('axes',num2str(jj+13))))
            cla(handles.(strcat('axes',num2str(jj+18))))
end
plot_all_graphs(hObject,handles,handles.plot_new)
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function save_menu_Callback(hObject, eventdata, handles)


% ------------------------------------------------------------------
--
function angles_save_txt_menu_Callback(hObject, eventdata, handles)
%
Q = handles.result.q;
[m,n] = size(Q);
Qtext =
[Q(1:6,:);Q(14:17,:);Q(7:12,:);Q(19:22,:);Q(13,:);zeros(1,n);Q(18,:)
];
[file,path] = uiputfile('*.txt','Save Joint Angles as');
DLMWRITE(file,Qtext,'delimiter','\t','precision','%.6f')
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function angles_save_csv_menu_Callback(hObject, eventdata, handles)

Q = handles.result.q;
[m,n] = size(Q);
Qtext =
[Q(1:6,:);Q(14:17,:);Q(7:12,:);Q(19:22,:);Q(13,:);zeros(1,n);Q(18,:)
```

```
];
[file,path] = uiputfile('*.csv','Save Joint Angles as');
csvid = fopen(file, 'w');

fprintf(csvid, '%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f \n', Qtext');

fclose(csvid);
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function ang_vel_acc_save_txt_menu_Callback(hObject, eventdata,
handles)
Q = handles.result.q;
dQ = handles.result.dq;
ddQ = handles.result.ddq;
[m,n] = size(Q);
Qtext =
[Q(1:6,:);Q(14:17,:);Q(7:12,:);Q(19:22,:);Q(13,:);zeros(1,n);Q(18,:)
];
dQtext =
[dQ(1:6,:);dQ(14:17,:);dQ(7:12,:);dQ(19:22,:);dQ(13,:);zeros(1,n);dQ
(18,:)];
ddQtext =
[ddQ(1:6,:);ddQ(14:17,:);ddQ(7:12,:);ddQ(19:22,:);ddQ(13,:);zeros(1,
n);ddQ(18,:)];


[file1,path] = uiputfile('*.txt','Save Joint Angles as');
DLMWRITE(file1,Qtext,'delimiter','\t','precision','%.6f')
[file2,path] = uiputfile('*.txt','Save Joint Velocities as');
DLMWRITE(file2,dQtext,'delimiter','\t','precision','%.6f')
[file3,path] = uiputfile('*.txt','Save Joint Accelerations as');
DLMWRITE(file3,ddQtext,'delimiter','\t','precision','%.6f')
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function ang_vel_acc_save_csv_menu_Callback(hObject, eventdata,
handles)
Q = handles.result.q;
dQ = handles.result.dq;
ddQ = handles.result.ddq;
[m,n] = size(Q);
Qtext =
[Q(1:6,:);Q(14:17,:);Q(7:12,:);Q(19:22,:);Q(13,:);zeros(1,n);Q(18,:)
];
dQtext =
[dQ(1:6,:);dQ(14:17,:);dQ(7:12,:);dQ(19:22,:);dQ(13,:);zeros(1,n);dQ
(18,:)];
ddQtext =
[ddQ(1:6,:);ddQ(14:17,:);ddQ(7:12,:);ddQ(19:22,:);ddQ(13,:);zeros(1,
n);ddQ(18,:)];

[file1,path] = uiputfile('*.csv','Save Joint Angles as');
```

```
csvid = fopen(file1, 'w');

fprintf(csvid, '%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f \n', Qtext');

fclose(csvid);
[file2,path] = uiputfile('*.csv','Save Joint Velocities as');
csvid = fopen(file2, 'w');

fprintf(csvid, '%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f \n', dQtext');

fclose(csvid);
[file3,path] = uiputfile('*.csv','Save Joint Accelerations as');
csvid = fopen(file3, 'w');

fprintf(csvid, '%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f,
%1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f, %1.4f \n', ddQtext');

fclose(csvid);


% ------------------------------------------------------------------
--
function joint_oper_menu_Callback(hObject, eventdata, handles)
Data_Hoap = handles.result;
[file1,path] = uiputfile('*.mat','Save Result as');
save(file1,Data_Hoap);
guidata(hObject,handles)


% ------------------------------------------------------------------
--
function control_menu_Callback(hObject, eventdata, handles)
steps_control
guidata(hObject,handles)
close(handles.figure1)

% --------------------------------------------------------------------
Callbacks
```

---

**ds_ss_step_hoap.m**

```
function [q,dq,ddq,trajectory,d_trajectory,dd_trajectory] =
ds_ss_step_hoap(delta,data,leg)
%
% DS_SS_STEP_HOAP FUNCTION
%
%
*********************************************************************
*****
global h hoap3 pvia
```

```
% INPUTS -------------------------------------------------------------
INPUTS
% Steps Data

L = data.L;
H = data.H;
q0 = data.q0;
% Time parameters
Ts = data.TS;
t0 = data.t0;
T = data.T;

delta_ss_com = [delta.delta_CoM_SS1 delta.delta_CoM_SS2];
delta_ss_ff = [delta.delta_FF_SS1 delta.delta_FF_SS2];

delta_RH = delta.delta_RH;
delta_LH = delta.delta_LH;

% Create Trajectory

humanoid_fields = humanoid_operational_fields ();

trajectory = create_trajectory_template (humanoid_fields, Ts);
d_trajectory = create_trajectory_template (humanoid_fields, Ts);
dd_trajectory = create_trajectory_template (humanoid_fields, Ts);

switch data.DS_or_SS

    case 'Double & Simple'

        alpha_ds = data.alpha_ds;
        alpha_sf = data.alpha_sf;
        T_ds = round_to_Ts(alpha_ds * T, Ts);
        T_ss = round_to_Ts((1-alpha_ds) * T, Ts);
        % Delta Data
        delta_ds_com = delta.delta_CoM_DS;

        switch leg

            case 'Right Leg Support' % Right Leg

                % Initial positions
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RF(q0)), Ts, t0),
'RF');

                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RH(q0)), Ts, t0),
'RH');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LH(q0)), Ts, t0),
'LH');


                [trajectory, d_trajectory, dd_trajectory] =
```

```
move_double_support (delta_ds_com, Ts, [t0;T_ds], trajectory,
d_trajectory, dd_trajectory,delta.interpola_CoM_DS);

                [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_ss_com, delta_ss_ff, Ts,
[T_ds;T_ds+T_ss/2;T], trajectory, d_trajectory, dd_trajectory,
'Simple Support RF',delta.interpola_CoM_SS,delta.interpola_FF_SS); %
Support on Right foot
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('RH',delta_RH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_RH);
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('LH',delta_LH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_LH);

                [q, dq, ddq] = inverse_right_ds_ss_hoap(q0,
trajectory, d_trajectory, dd_trajectory, h);

            case 'Left Leg Support' % Left Leg


                % Initial positions

                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LF(q0)), Ts, t0),
'LF');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RH(q0)), Ts, t0),
'RH');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LH(q0)), Ts, t0),
'LH');



                [trajectory, d_trajectory, dd_trajectory] =
move_double_support (delta_ds_com, Ts, [t0;T_ds], trajectory,
d_trajectory, dd_trajectory,delta.interpola_CoM_DS);

                [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_ss_com, delta_ss_ff, Ts,
[T_ds;T_ds+T_ss/2;T], trajectory, d_trajectory,
dd_trajectory,'Simple Support
LF',delta.interpola_CoM_SS,delta.interpola_FF_SS);

                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('RH',delta_RH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_RH);
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('LH',delta_LH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_LH);

                [q, dq, ddq] = inverse_left_ds_ss_hoap(q0,
trajectory, d_trajectory, dd_trajectory, h);
```

```
        end

    case 'Simple'
        alpha_ds = 0;
        alpha_sf = data.alpha_sf;
        T_ds = 0;
        T_ss = T;

        delta_ds_com = zeros(6,1);

        switch leg % In Simple SUPPORT

            case 'Right Leg Support' % Right Leg

                % Initial positions
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RF(q0)), Ts, t0),
'RF');
trajectory = insert_trajectory(trajectory, humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RH(q0)), Ts, t0),
'RH');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LH(q0)), Ts, t0),
'LH');

                [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_ss_com, delta_ss_ff, Ts,
[T_ds;T_ds+T_ss/2;T], trajectory, d_trajectory, dd_trajectory,
'Simple Support RF',delta.interpola_CoM_SS,delta.interpola_FF_SS); %
Support on Right foot
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('RH',delta_RH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_RH);
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('LH',delta_LH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_LH);

                [q, dq, ddq] = inverse_right_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h);

            case 'Left Leg Support' % Left Leg

                % Initial positions

                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LF(q0)), Ts, t0),
'LF');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_RH(q0)), Ts, t0),
'RH');
                trajectory = insert_trajectory(trajectory,
humanoid_fields,
create_trajectory_structure(pose_quat2rpy(h.CoM_T_LH(q0)), Ts, t0),
'LH');
```

```
                [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_ss_com, delta_ss_ff, Ts,
[T_ds;T_ds+T_ss/2;T], trajectory, d_trajectory,
dd_trajectory,'Simple Support
LF',delta.interpola_CoM_SS,delta.interpola_FF_SS);

                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('RH',delta_RH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_RH);
                [trajectory, d_trajectory, dd_trajectory] =
moving_arm ('LH',delta_LH,zeros(6,1),zeros(6,1),Ts, [t0 T],
trajectory, d_trajectory, dd_trajectory,delta.interpola_LH);

                [q, dq, ddq] = inverse_left_ss_hoap(q0, trajectory,
d_trajectory, dd_trajectory, h);


        end

end
```

---

move_double_support.m ----------

```
function [trajectory, d_trajectory, dd_trajectory] =
move_double_support (delta_p, Ts, T, trajectory, d_trajectory,
dd_trajectory,interpola)
ZERO_CON = zeros(6,1);
humanoid_fields = humanoid_operational_fields ();
% L = round(T(1)/Ts)+1;

L = size(trajectory.CoM,2);
% Final Positions
p0_com = trajectory.CoM(:, L);
p1_com = p0_com + delta_p;

% Generating foot trajectory
P0 = set_trajectory_condition(T(1), p0_com, ZERO_CON, ZERO_CON);
P1 = set_trajectory_condition(T(2), p1_com, ZERO_CON, ZERO_CON);
% Ahora seleccionamos la trayectoria a seguir

switch interpola
    case 'Linear'
        [pp_com1, dpp_com1, ddpp_com1] = lineartrajectory(P0, P1,
Ts);
        pp_com  = create_trajectory_structure(pp_com1,  Ts, T);
        dpp_com = create_trajectory_structure(dpp_com1, Ts, T);
        ddpp_com = create_trajectory_structure(ddpp_com1, Ts, T);
    case 'Polinomic'
        % Polinomic 3
        [pp_com, dpp_com, ddpp_com] = polinomic3_trajectory (P0, P1,
Ts);

    case 'Spline'
        % Spline
        [pp_com, dpp_com, ddpp_com] = spline_interpolation (P0, P1,
```

```
Ts);



end
% AquÌ cambio el tiempo que le metemos a la funciÛn
create_trajectory_structure
%ya que es tiempo inicial y tiempo final


trajectory   = insert_trajectory(trajectory,   humanoid_fields,
pp_com,  'CoM');
d_trajectory = insert_trajectory(d_trajectory, humanoid_fields,
dpp_com, 'CoM');
dd_trajectory = insert_trajectory(dd_trajectory, humanoid_fields,
ddpp_com, 'CoM');
end
```

move_simple_support.m ----------

```
function [trajectory, d_trajectory, dd_trajectory] =
move_simple_support (delta_com, delta_FF, Ts, T, trajectory,
d_trajectory, dd_trajectory, leg,interpolaCoM,interpolaFF)
humanoid_fields = humanoid_operational_fields ();

% Setting values
ZERO_CON = zeros(6,1);

% Trajectory for the CoM
n = round(T(1)/Ts)+1;
p0_com = trajectory.CoM(:,n);
p1_com = p0_com + delta_com(:,1);
p2_com = p0_com + delta_com(:,2);

P0_com_s = set_trajectory_condition(T(1), p0_com, ZERO_CON,
ZERO_CON);
P1_com_s = set_trajectory_condition(T(2), p1_com,
evaluate_mean_velocity (delta_com, T), ZERO_CON);
P2_com_s = set_trajectory_condition(T(3), p2_com, ZERO_CON,
ZERO_CON);

switch interpolaCoM
    case 'Linear'
        [pp1_com, dpp1_com, ddpp1_com] = lineartrajectory(P0_com_s,
P1_com_s, Ts);
        [pp2_com, dpp2_com, ddpp2_com] = lineartrajectory(P0_com_s,
P1_com_s, Ts);
    case 'Polinomic'
        % Polinomic 3
        [pp1_com, dpp1_com, ddpp1_com] = polinomic3_trajectory
(P0_com_s, P1_com_s, Ts);
        [pp2_com, dpp2_com, ddpp2_com] = polinomic3_trajectory
(P1_com_s, P2_com_s, Ts);
    case 'Spline'
        % Spline
        [pp1_com, dpp1_com, ddpp1_com] = spline_interpolation
(P0_com_s, P1_com_s, Ts);
        [pp2_com, dpp2_com, ddpp2_com] = spline_interpolation
(P1_com_s, P2_com_s, Ts);
```

```
end



pp_com1  = combine_trajectories(pp1_com,  pp2_com);
dpp_com1 = combine_trajectories(dpp1_com, dpp2_com);
ddpp_com1 = combine_trajectories(ddpp1_com, ddpp2_com);

pp_com  = create_trajectory_structure(pp_com1.data,  Ts, [T(1)
T(3)]);
dpp_com = create_trajectory_structure(dpp_com1.data, Ts, [T(1)
T(3)]);
ddpp_com = create_trajectory_structure(ddpp_com1.data, Ts, [T(1)
T(3)]);

trajectory   = insert_trajectory(trajectory,   humanoid_fields,
pp_com,  'CoM');
d_trajectory = insert_trajectory(d_trajectory, humanoid_fields,
dpp_com, 'CoM');
dd_trajectory = insert_trajectory(dd_trajectory, humanoid_fields,
ddpp_com, 'CoM');

% Trajectory for the Floating Foot
if strcmp(leg,'Simple Support LF')        %Support on Left Foot
    FF = 'RF';
    foot =1;
elseif strcmp(leg,'Simple Support RF')               %Support on
Right Foot
    FF = 'LF';
    foot =-1;
end

p0_FF = trajectory.(FF)(:,n);
p1_FF = p0_FF + delta_FF(:,1);
p2_FF = p0_FF + delta_FF(:,2);

% Generating flotating foot trajectory
P0_FF_s = set_trajectory_condition(T(1), p0_FF, ZERO_CON, ZERO_CON);
P1_FF_s = set_trajectory_condition(T(2), p1_FF,
evaluate_mean_velocity (delta_FF, T), ZERO_CON);
P2_FF_s = set_trajectory_condition(T(3), p2_FF, ZERO_CON, ZERO_CON);

switch interpolaFF
    case 'Linear'
        [pp1_FF, dpp1_FF, ddpp1_FF] = lineartrajectory(P0_com_s,
P1_com_s, Ts);
        [pp2_FF, dpp2_FF, ddpp2_FF] = lineartrajectory(P0_com_s,
P1_com_s, Ts);
    case 'Polinomic'
        % Polinomic 3
        [pp1_FF, dpp1_FF, ddpp1_FF] = polinomic3_trajectory
(P0_FF_s, P1_FF_s, Ts);
        [pp2_FF, dpp2_FF, ddpp2_FF] = polinomic3_trajectory
(P1_FF_s, P2_FF_s, Ts);
    case 'Spline'
        % Spline
        [pp1_FF, dpp1_FF, ddpp1_FF] = spline_interpolation (P0_FF_s,
P1_FF_s, Ts);
        [pp2_FF, dpp2_FF, ddpp2_FF] = spline_interpolation (P1_FF_s,
```

```
P2_FF_s, Ts);
end

pp_FF1 = combine_trajectories(pp1_FF, pp2_FF);
dpp_FF1 = combine_trajectories(dpp1_FF, dpp2_FF);
ddpp_FF1 = combine_trajectories(ddpp1_FF, ddpp2_FF);

pp_FF  = create_trajectory_structure(pp_FF1.data,  Ts, [T(1) T(3)]);
dpp_FF = create_trajectory_structure(dpp_FF1.data, Ts, [T(1) T(3)]);
ddpp_FF = create_trajectory_structure(ddpp_FF1.data, Ts, [T(1)
T(3)]);
pp_SF  = create_trajectory_structure(foot*ones(size(pp_FF1.time)),
Ts, [T(1) T(3)]);

d_trajectory = insert_trajectory(d_trajectory, humanoid_fields,
pp_SF,  'SF');
dd_trajectory = insert_trajectory(dd_trajectory, humanoid_fields,
pp_SF,  'SF');
trajectory   = insert_trajectory(trajectory,   humanoid_fields,
pp_SF,  'SF');

trajectory   = insert_trajectory(trajectory,   humanoid_fields,
pp_FF,  FF);
d_trajectory = insert_trajectory(d_trajectory, humanoid_fields,
dpp_FF, FF);
dd_trajectory = insert_trajectory(dd_trajectory, humanoid_fields,
ddpp_FF, FF);
end

function vel_m = evaluate_mean_velocity (delta, T)
vel1 = delta(:,1)/(T(2)-T(1));
vel2 = (delta(:,2)-delta(:,1))/(T(3)-T(2));
vel_m = 0.5*vel1+0.5*vel2;
end
```

moving_arm.m ----------

```
function [traj, d_traj, dd_traj] = moving_arm
(manipulator,delta_A,d_delta_A,dd_delta_A,Ts, T, traj, d_traj,
dd_traj,interpola)
global p_via

humanoid_fields = humanoid_operational_fields ();

% Trajectory for the Arm
% p : position
% a : orientation

n = round(T(1)/Ts)+1;
p0_A = traj.(manipulator)(:,n);
p1_A = p0_A + delta_A(:,1);


dp0_A = d_traj.(manipulator)(:,n);
dp1_A = dp0_A + d_delta_A(:,1);

ddp0_A = dd_traj.(manipulator)(:,n);
ddp1_A = ddp0_A + dd_delta_A(:,1);
```

```
% set trajectory for position

pvia = [p_via;zeros(3,1)];

Pvia = set_trajectory_condition(round(mean(T)/Ts)*Ts, pvia,
zeros(6,1), zeros(6,1));

P0_A_s = set_trajectory_condition(T(1), p0_A, dp0_A, ddp0_A);
P1_A_s = set_trajectory_condition(T(2), p1_A, dp1_A, ddp1_A);

switch interpola
    case 'Linear'
        [pp_com1, dpp_com1, ddpp_com1] = lineartrajectory(P0_A_s,
P1_A_s, Ts);
        pp_A1  = create_trajectory_structure(pp_com1,  Ts, T);
        dpp_A1= create_trajectory_structure(dpp_com1, Ts, T);
        ddpp_A1= create_trajectory_structure(ddpp_com1, Ts, T);
    case 'Ciruclar'
        [pp_A1, dpp_A1, ddpp_A1, C, r, plane, n] =
arc_trajectory3(P0_A_s,Pvia,P1_A_s,Ts);
    case 'Polinomic'
        [pp_A1, dpp_A1, ddpp_A1] = polinomic3_trajectory(P0_A_s,
P1_A_s, Ts);
    case 'Spline'

        [pp_A1, dpp_A1, ddpp_A1] = spline_interpolation (P0_A_s,
P1_A_s, Ts);

end

traj   = insert_trajectory(traj,   humanoid_fields, pp_A1,
manipulator);
d_traj = insert_trajectory(d_traj, humanoid_fields, dpp_A1,
manipulator);
dd_traj = insert_trajectory(dd_traj, humanoid_fields, ddpp_A1,
manipulator);


end
```

# Funciones gráficas

### Función *plot_movies*

```
function plot_movies(data,colord,colname)

if nargin < 2
    colord = 1;
else
```

```
end
colordata = {'b','m'};
%
**********************************************************************
*****
k = length(data);


plot3(data(1,:),data(2,:),data(3,:),'-','DisplayName', colname);

xlabel('y','FontSize',12);
ylabel('x','FontSize',12);
zlabel('z','FontSize',12);
axis vis3d square;
view([40,30]);
camlight;
grid on;


l = 0.05*max([abs((data(1,k)-data(1,1))),abs((data(2,k)-
data(2,1))),abs((data(3,k)-data(3,1)))]);

lx =l;ly = l; lz = l;

hold on;
plot3(data(1,k),data(2,k),data(3,k),'or','MarkerFaceColor','r','Mark
erSize',8,'DisplayName', colname);
hold
on;plot3(data(1,1),data(2,1),data(3,1),'og','MarkerFaceColor','g','M
arkerSize',8,'DisplayName', colname);
hold on
text(data(1,1),data(2,1),data(3,1),texlabel(strcat('Initial
',colname)),'DisplayName', colname);
text(data(1,k),data(2,k),data(3,k),texlabel(strcat('Final
',colname)),'DisplayName', colname);

COM = hgtransform;% S
COM2 = hgtransform('Parent',COM,'DisplayName', colname);

h1 = link(lx,ly,lz,colord);
h2 = link(-lx,ly,lz,colord);
h3 = link(-lx,-ly,lz,colord);
h4 = link(lx,-ly,lz,colord);
h5 = link(lx,ly,-lz,colord);
h6 = link(-lx,ly,-lz,colord);
h7 = link(-lx,-ly,-lz,colord);
h8 = link(lx,-ly,-lz,colord);

set(h1,'Parent',COM2);
set(h2,'Parent',COM2);
set(h3,'Parent',COM2);
set(h4,'Parent',COM2);
set(h5,'Parent',COM2);
set(h6,'Parent',COM2);
set(h7,'Parent',COM2);
set(h8,'Parent',COM2);
MP = zeros(4,4);

for i=1:10:k-1
```

```
   MP = makehgtform('translate',[data(1,i) data(2,i)
data(3,i)])*makehgtform('zrotate',data(6,i))*makehgtform('yrotate',d
ata(5,i))*makehgtform('xrotate',data(4,i));
   set(COM2,'Matrix',MP);
   axis vis3d square
   drawnow
end


end
```

## Función *prueba_ZMP*

```
function prueba_ZMP(handles)
% ******************
%
%
% Block specification
Lx = 0.073;
Ly = 0.04+0.068;
Lz = 0.01;

% Motion data
t = [0:0.002:1]';        % Time data
r = [0*t, 0*t, 0*t];     % Position data
A = [0*t, 0*t, 2*pi*t]; % Orientation data (x-y-z Euler angle)

n_time=length(t);

% Compute propagation of vertices and patches
for i_time=1:n_time

    R = Euler2R(A(i_time,:));

    VertexData(:,:,i_time) =
GeoVerMakeBlock(r(i_time,:),R,[Lx,Ly,Lz]);
    PatchData(:,:,:,i_time) =
GeoPatMakeBlock(VertexData(:,:,i_time));
end

n_pat = size(PatchData,3);

% Draw initial figure
for i_pat=1:n_pat

    XData = PatchData(:,1,i_pat,1);
    YData = PatchData(:,2,i_pat,1);
    ZData = PatchData(:,3,i_pat,1);


    hLF(i_pat) = patch(XData,YData,ZData,[0.6 0.6 0.6]);

    set(hLF(i_pat),'FaceLighting','phong','EdgeLighting','phong');
    set(hLF(i_pat),'EraseMode','normal');
end
% Compute propagation of vertices and patches
for i_time=1:n_time
```

```
    R = Euler2R(A(i_time,:));

    VertexData(:,:,i_time) = GeoVerMakeBlock(r(i_time,:),R,[-
Lx,Ly,Lz]);
    PatchData(:,:,:,i_time) =
GeoPatMakeBlock(VertexData(:,:,i_time));
end
% Draw initial figure
for i_pat=1:n_pat

    XData = PatchData(:,1,i_pat,1);
    YData = PatchData(:,2,i_pat,1);
    ZData = PatchData(:,3,i_pat,1);


    hRF(i_pat) = patch(XData,YData,ZData,[1 0.6 0.6]);

    set(hRF(i_pat),'FaceLighting','phong','EdgeLighting','phong');
    set(hRF(i_pat),'EraseMode','normal');
end

xlabel('y','FontSize',14);
ylabel('x','FontSize',14);
zlabel('z','FontSize',14);
set(gca,'FontSize',14,'XDir','reverse');
axis vis3d square;
view([40,30]);
camlight;
grid on;
xlim([-0.2,0.3]);
ylim([-0.2,0.2]);
zlim([-0.4,0.1]);
COM = hgtransform;% S
COM2 = hgtransform('Parent',COM);
COM3 = hgtransform('Parent',COM);
set(hLF,'Parent',COM2);
set(hRF,'Parent',COM3);
set(COM2,'Matrix',makehgtform('translate',[(0.038-0.0315) -0.04
0]));
set(COM3,'Matrix',makehgtform('translate',[(-0.038+0.0315) -0.04
0]));


MLF = zeros(4,4);
MRF = zeros(4,4);
MCoM = zeros(4,4);
data.CoM = handles.result.trajectory.CoM;
k = length(data.CoM);
tk = round(0.01*k);
if strcmp(handles.Input_data.Leg,'Right Leg Support')
    data.S = handles.result.trajectory.RF;
    data.F = handles.result.trajectory.LF;
    hold on;plot3((0.038+data.F(2,1)),(data.F(1,1)),...
        (data.F(3,1)+data.S(3,1)),...
        'og','MarkerFaceColor','g','MarkerSize',7);
    hold on;plot3(data.CoM(2,1),data.CoM(1,1),data.CoM(3,1),...
        'og','MarkerFaceColor','g','MarkerSize',7);
    text((data.S(2,1)-0.038),(data.S(1,1)-Ly),data.S(3,1),...
```

```
        texlabel('Right Foot'),'EdgeColor','blue','LineWidth',3,...
        'HorizontalAlignment','center')
    text((0.038+0.25*Lx+data.F(2,1)),(data.F(1,1)-Ly),...
        (data.F(3,1)+data.S(3,1)),texlabel('Left
Foot'),'EdgeColor',...
        'blue','LineWidth',3,'HorizontalAlignment','center')
    text(data.CoM(2,k),data.CoM(1,k),(data.CoM(3,k)+Lx),...
        texlabel('CoM'),'EdgeColor','red','LineWidth',3,...
        'HorizontalAlignment','center');
    for i=tk:tk:k-1
       MLF = makehgtform('translate',...
           [(0.038-0.0315+data.F(2,i)) (data.F(1,i)-0.04)
(data.F(3,i)+data.S(3,i))])*makehgtform('zrotate',data.F(6,i))*makeh
gtform('xrotate',data.F(5,i))*makehgtform('yrotate',data.F(4,i));
       MRF = makehgtform('translate',...
           [(0.0315-0.038) -0.04
(data.S(3,i))])*makehgtform('zrotate',data.S(6,i))*makehgtform('xrot
ate',data.S(5,i))*makehgtform('yrotate',data.S(4,i));

       set(COM2,'Matrix',MLF);
       set(COM3,'Matrix',MRF);
       hold on;plot3((0.038+data.F(2,i)),(data.F(1,i)),...
           (data.F(3,i)+data.S(3,i)),'ob','MarkerFaceColor',...
           'b','MarkerSize',5);
       hold
on;plot3(data.CoM(2,i),data.CoM(1,i),data.CoM(3,i),'ok',...
           'MarkerFaceColor','k','MarkerSize',5);
       axis square

       drawnow

    end
    i=k;
    MLF = makehgtform('translate',...
        [(0.038-0.0315+data.F(2,i)) (data.F(1,i)-0.04)
(data.F(3,i)+data.S(3,i))])*makehgtform('zrotate',data.F(6,i))*makeh
gtform('xrotate',data.F(5,i))*makehgtform('yrotate',data.F(4,i));
    MRF = makehgtform('translate',...
        [(0.0315-0.038) -0.04
(data.S(3,i))])*makehgtform('zrotate',data.S(6,i))*makehgtform('xrot
ate',data.S(5,i))*makehgtform('yrotate',data.S(4,i));

    set(COM2,'Matrix',MLF);
    set(COM3,'Matrix',MRF);

    hold on;plot3((0.038+data.F(2,k)),(data.F(1,k)),...
        (data.F(3,k)+data.S(3,k)),'or','MarkerFaceColor',...
        'r','MarkerSize',7);
    hold on;plot3(data.CoM(2,k),data.CoM(1,k),data.CoM(3,k),'or',...
        'MarkerFaceColor','r','MarkerSize',7);
    axis vis3d square

    drawnow
else
    data.S = handles.result.trajectory.LF;
    data.F = handles.result.trajectory.RF;
    hold on;plot3((-0.038+data.F(2,1)),(data.F(1,1)),...
        (data.F(3,1)+data.S(3,1)),'og','MarkerFaceColor',...
        'g','MarkerSize',7);
    hold on;plot3(data.CoM(2,1),data.CoM(1,1),data.CoM(3,1),'og',...
```

```
        'MarkerFaceColor','g','MarkerSize',7);
    text((data.F(2,1)-0.038),(data.F(1,1)-
Ly),(data.F(3,1)+data.S(3,1)),...
        texlabel('Right Foot'),'EdgeColor','blue','LineWidth',3,...
        'HorizontalAlignment','center')
    text((0.038+0.25*Lx+data.S(2,1)),(data.S(1,1)-
Ly),data.S(3,1),...
        texlabel('Left Foot'),'EdgeColor','blue','LineWidth',3,...
        'HorizontalAlignment','center')
    text(data.CoM(2,k),data.CoM(1,k),(data.CoM(3,k)+Lx),...
        texlabel('CoM'),'EdgeColor','red','LineWidth',3,...
        'HorizontalAlignment','center');
    for i=tk:tk:k-1
       MRF = makehgtform('translate',...
           [(0.0315-0.038+data.F(2,i)) (data.F(1,i)-0.04)
(data.F(3,i)+data.S(3,i))])*makehgtform('zrotate',data.F(6,i))*makeh
gtform('xrotate',data.F(5,i))*makehgtform('yrotate',data.F(4,i));
       MLF = makehgtform('translate',...
           [(0.038-0.0315) -0.04
(data.S(3,i))])*makehgtform('zrotate',data.S(6,i))*makehgtform('xrot
ate',data.S(5,i))*makehgtform('yrotate',data.S(4,i));

       set(COM2,'Matrix',MLF);
       set(COM3,'Matrix',MRF);
       hold on;plot3((-0.038+data.F(2,i)),(data.F(1,i)),...
           (data.F(3,i)+data.S(3,i)),'ob','MarkerFaceColor',...
           'b','MarkerSize',5);
       hold
on;plot3(data.CoM(2,i),data.CoM(1,i),data.CoM(3,i),'ok',...
           'MarkerFaceColor','k','MarkerSize',5);
       axis vis3d square

       drawnow
    end
    i=k;
    MRF = makehgtform('translate',...
        [(0.0315-0.038+data.F(2,i)) (data.F(1,i)-0.04)
(data.F(3,i)+data.S(3,i))])*makehgtform('zrotate',data.F(6,i))*makeh
gtform('xrotate',data.F(5,i))*makehgtform('yrotate',data.F(4,i));
    MLF = makehgtform('translate',...
        [(0.038-0.0315) -0.04
(data.S(3,i))])*makehgtform('zrotate',data.S(6,i))*makehgtform('xrot
ate',data.S(5,i))*makehgtform('yrotate',data.S(4,i));%*makehgtform('
translate',[data.S(2,i) data.S(1,i) data.S(3,i)]);

    set(COM2,'Matrix',MLF);
    set(COM3,'Matrix',MRF);
    hold on;plot3((-0.038+data.F(2,k)),(data.F(1,k)),...

(data.F(3,k)+data.S(3,k)),'or','MarkerFaceColor','r','MarkerSize',7)
;
    hold on;plot3(data.CoM(2,k),data.CoM(1,k),data.CoM(3,k),...
        'or','MarkerFaceColor','r','MarkerSize',7);
    axis vis3d square

    drawnow
end
view(2)
```

## Función Euler2R

```
function R = Euler2R(A)

% Euler angle -> Orientation matrix
a1 = A(1);
a2 = A(2);
a3 = A(3);

R1 = [1, 0, 0;
    0, cos(a1), -sin(a1);
    0, sin(a1), cos(a1)];

R2 = [cos(a2), 0, sin(a2);
    0, 1, 0;
    -sin(a2), 0, cos(a2)];

R3 = [cos(a3), -sin(a3), 0;
    sin(a3), cos(a3), 0;
    0, 0, 1];

R = R1*R2*R3;
```

## Función GeoVerMakeBlock

```
function VertexData =
GeoVerMakeBlock(Location,Orientation,SideLength)

r = Location;
R = Orientation;

Lx = SideLength(1);
Ly = SideLength(2);
Lz = SideLength(3);

VertexData_0 = [Lx*ones(8,1), Ly*ones(8,1), Lz*ones(8,1)]...
    .*[0,0,0;
    1,0,0;
    0,1,0;
    0,0,1;
    1,1,0;
    0,1,1;
    1,0,1;
    1,1,1];

n_ver = 8;

for i_ver=1:n_ver

    VertexData(i_ver,:) = r + VertexData_0(i_ver,:)*R';
end
```

## Función GeoPatMakeBlock

```
function PatchData = GeoPatMakeBlock(VertexData)
```

```
Index_Patch = ...
    [1,2,5,3;
    1,3,6,4;
    1,4,7,2;
    4,7,8,6;
    2,5,8,7;
    3,6,8,5];

n_pat = 6;

for i_pat=1:n_pat

    XData = VertexData(Index_Patch(i_pat,:),1);
    YData = VertexData(Index_Patch(i_pat,:),2);
    ZData = VertexData(Index_Patch(i_pat,:),3);

    PatchData(:,:,i_pat) = [XData,YData,ZData];
end
```

# Anexo 2

# Manual de usuario de la interfaz

# Manual de usuario de la interfaz

La interfaz se podrá ejecutar en equipos dispongan de MATLAB o no. Es compatible con sistemas operativos Windows de 64 y 32 bits y con Mac OS, aunque hay que destacar que en estos últimos no hay ejecutables como tales y solo se podrá tener acceso a la interfaz a través del programa MATLAB para Mac OS.

Para equipos Windows con Matlab instalado será suficiente con ejecutar el archivo “Main_Hoap3.exe” que se encuentra en la carpeta del programa adjunto.

Para equipos Windows que no dispongan de Matlab hay que instalar el MCR (MATLAB Compiler Runtime) que viene adjunto para que la interfaz puede hacer uso de las librerías del programa.

# Manipulator Control

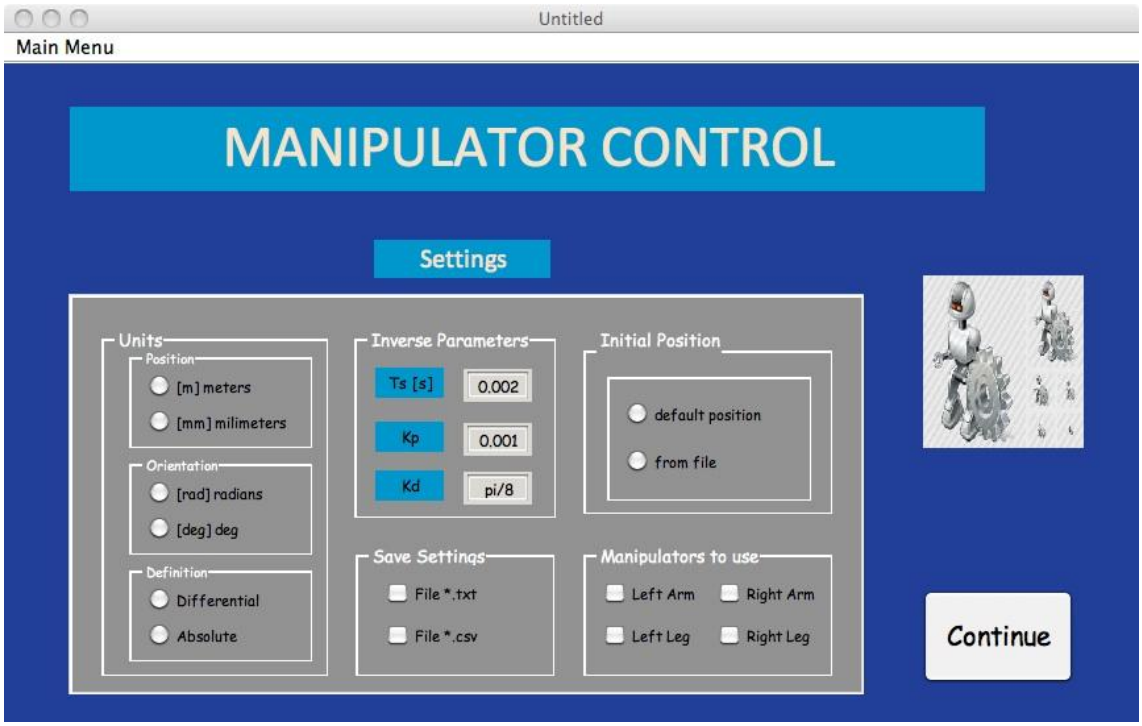
## Introducción al control del manipulador

El usuario podrá diseñar trayectorias para cada manipulador de manera aislada, como si se tratase de un <<laboratorio de trayectorias>>. Es un control de bajo nivel que permite estudiar individualmente la trayectoria en el espacio de trabajo del extremo de cada manipulador, realizar la inversión cinemática del manipulador aislado o realizarla de forma completa en el robot entero.

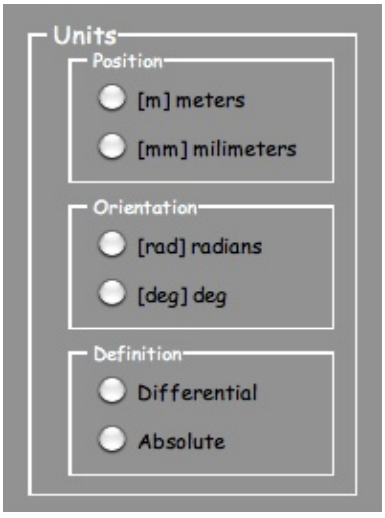
Se podrán generar dos trayectorias consecutivas, estudiar la evolución en el tiempo de cada coordenada de posición y orientación, y simular el movimiento del extremo del manipulador en el expacio.

## Configuración de parámetros

La primera ventana con la que se encontrará el usuario será la de configuración de los parámetros del subprograma Manipulator Control.

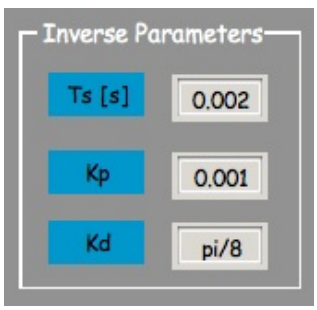



Pantalla inicial del subprograma Manipulator Control




En primer lugar hay que indicar las unidades de posición (metros o milímetros) y las unidades de orientación (radianes o grados). La definición de las localizaciones podrá ser diferencial (Differential) o absoluta. La localización diferencial permite trabajar con incrementos de posición y orientación en caso de que no se tenga información en el espacio cartesiano con coordenadas absolutas respecto al sistema situado en el centro de masas.

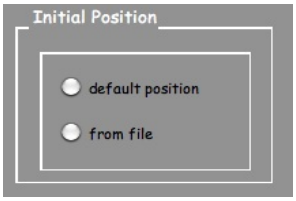
Solo se podrá marcar una de las dos opciones al tratarse de <<radio buttons>>.




El paso para la generación de la trayectoria es el tiempo Ts en segundos, y será el mismo para todos los manipuladores y trayectorias. Así mismo, se da la opción de controlar los parámetros de inversión del algoritmo de solución para

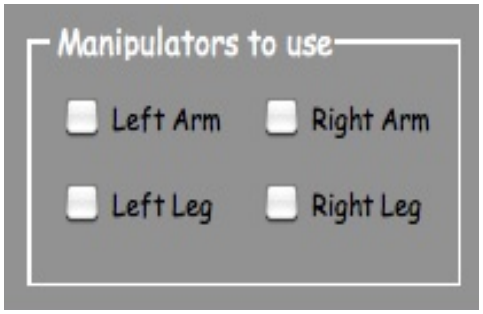
realizar un estudio algo más detallado de este tipo de realimentaciones.

La configuración inicial de los motores del robot se puede elegir entre una configuración por defecto o una configuración guardada en archivo con extensión csv con el siguiente formato comentado en el apartado de diseño de la interfaz descrito en la memoria.




| Índice qi | Articulación | Movimiento |
|---|---|---|
| q1 | RLEG_JOINT 1 | Right hip joint torsion |
| q2 | RLEG_JOINT 2 | Right hip joint roll |
| q3 | RLEG_JOINT 3 | Right hip joint pitch |
| q4 | RLEG_JOINT 4 | Right knee |
| q5 | RLEG_JOINT 5 | Right ankle pitch |
| q6 | RLEG_JOINT 6 | Right ankle roll |
| q7 | LLEG_JOINT 1 | Left hip joint torsion |
| q8 | LLEG_JOINT 2 | Left hip joint roll |
| q9 | LLEG_JOINT 3 | Left hip joint pith |
| q10 | LLEG_JOINT 4 | Left knee |
| q11 | LLEG_JOINT 5 | Left ankle pitch |
| q12 | LLEG_JOINT 6 | Left ankle roll |
| q13 | BODY_JOINT 1 | Waist pitch |
| q14 | RARM_JOINT 1 | Right shoulder pitch |
| q15 | RARM_JOINT 2 | Right shoulder roll |
| q16 | RARM_JOINT 3 | Right shoulder torsion |
| q17 | RARM_JOINT 4 | Right elbow |
| q18 | RARM_JOINT 5 | Right hand torsion |
| q19 | LARM_JOINT 1 | Left shoulder pitch |
| q20 | LARM_JOINT 2 | Left shoulder roll |
| q21 | LARM_JOINT 3 | Left shoulder torsion |
| q22 | LARM_JOINT 4 | Left elbow |
| q23 | LARM_JOINT 5 | Left hand torsion |

El programa cargará la última línea del archivo csv que se elija.

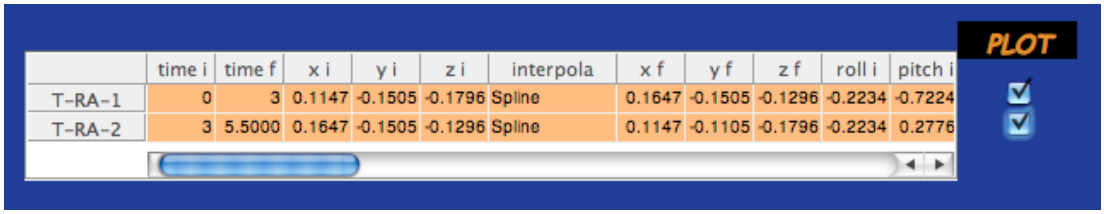



Se podrán elegir los manipuladores a estudiar: brazo derecho, brazo izquierdo, pierna izquierda y/o pierna derecha. A su vez, se podrá elegir entre los manipuladores correspondientes a la pierna de apoyo y a la pierna flotante. En el caso de la pierna soporte, la trayectoria generada será la del centro de masas. Esta circunstancia viene reflejada en el capitulo an de diseño de la interfaz.

Para validar las opciones marcadas pulse el botón y espere a que se cargue la ventana de generación de trayectorias para los manipuladores que haya elegido.


## Generación de trayectorias en el espacio de trabajo

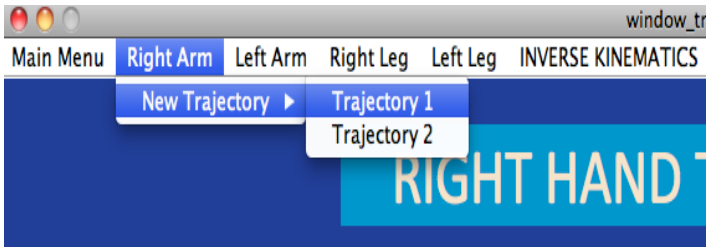

Validada la opción anterior, el programa muestra la pantalla de Manipulator Control. En ella vienen tabulados los datos correspondientes a la trayectoria tales como los tiempos inicial y final de cada trayectoria, las coordenadas de posición iniciales y finales, el tipo de interpolación empleada en la posición, la orientación inicial y final (roll, pitch, yaw), etc.



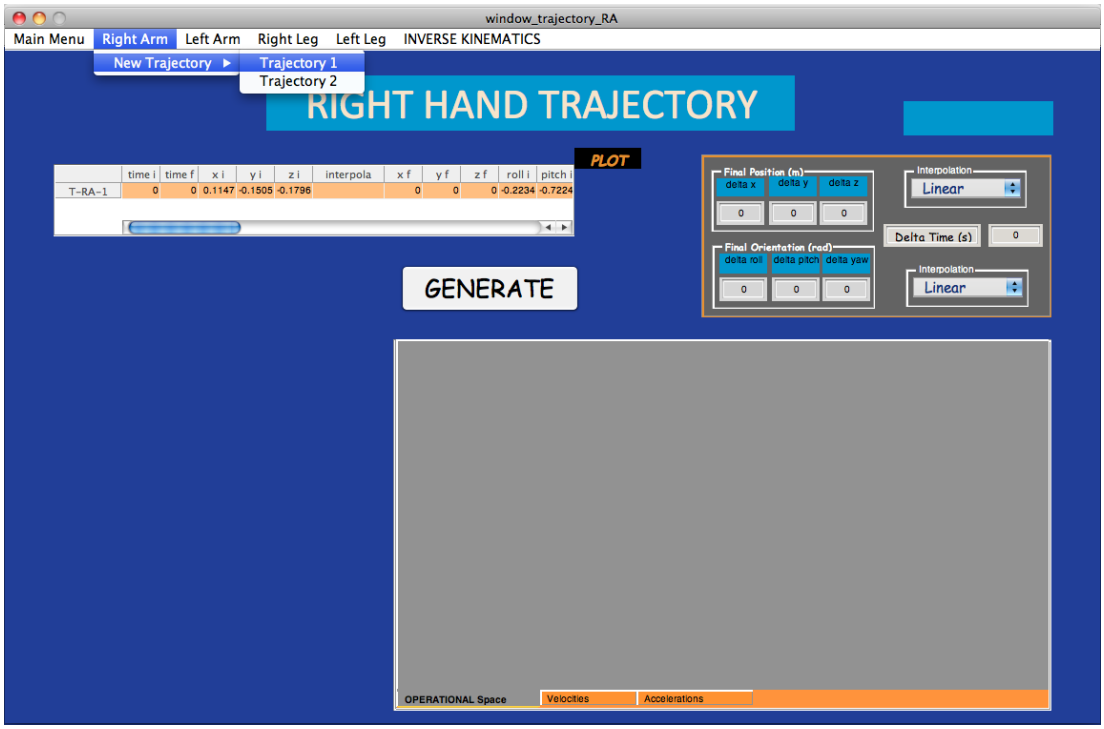

| | time i | time f | x i | y i | z i | interpola | x f | y f | z f | roll i | pitch i |
|---|---|---|---|---|---|---|---|---|---|---|---|
| T-RA-1 | 0 | 3 | 0.1147 | -0.1505 | -0.1796 | Spline | 0.1647 | -0.1505 | -0.1296 | -0.2234 | -0.7224 |
| T-RA-2 | 3 | 5.5000 | 0.1647 | -0.1505 | -0.1296 | Spline | 0.1147 | -0.1105 | -0.1796 | -0.2234 | 0.2776 |

Panel de datos de las trayectorias

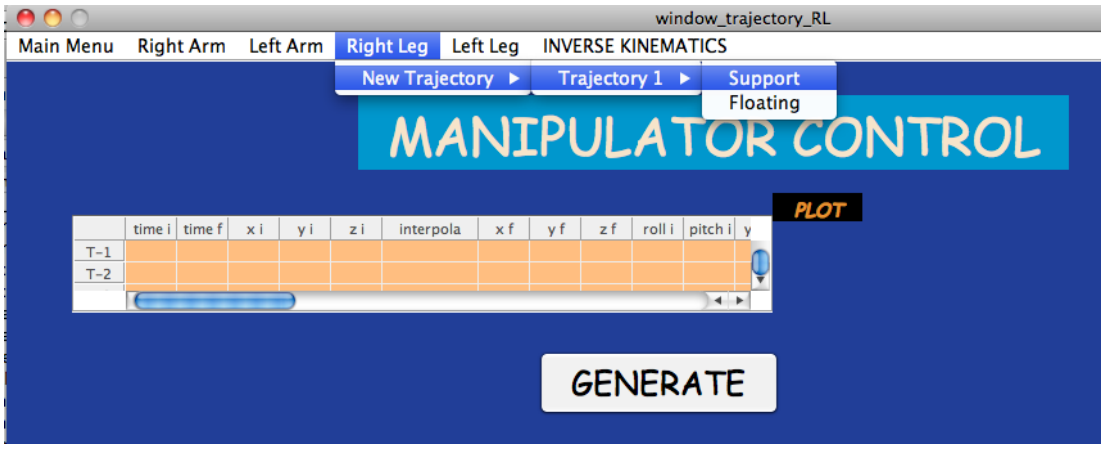
Además de proporcionar la opción de volver al menú principal, en esta pantalla podemos obtener una información más detallada del manipulador deseado pinchando en el menú correspondiente.




En dicho menú podemos seleccionar la primera trayectoria a generar.



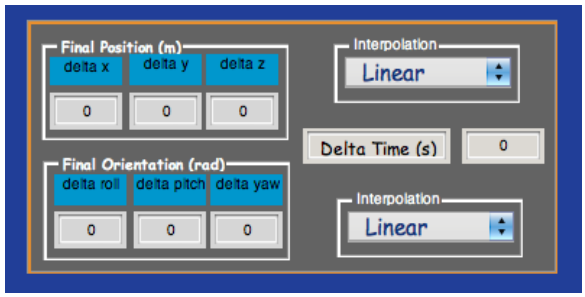

Menú desplegable de selección de trayectorias

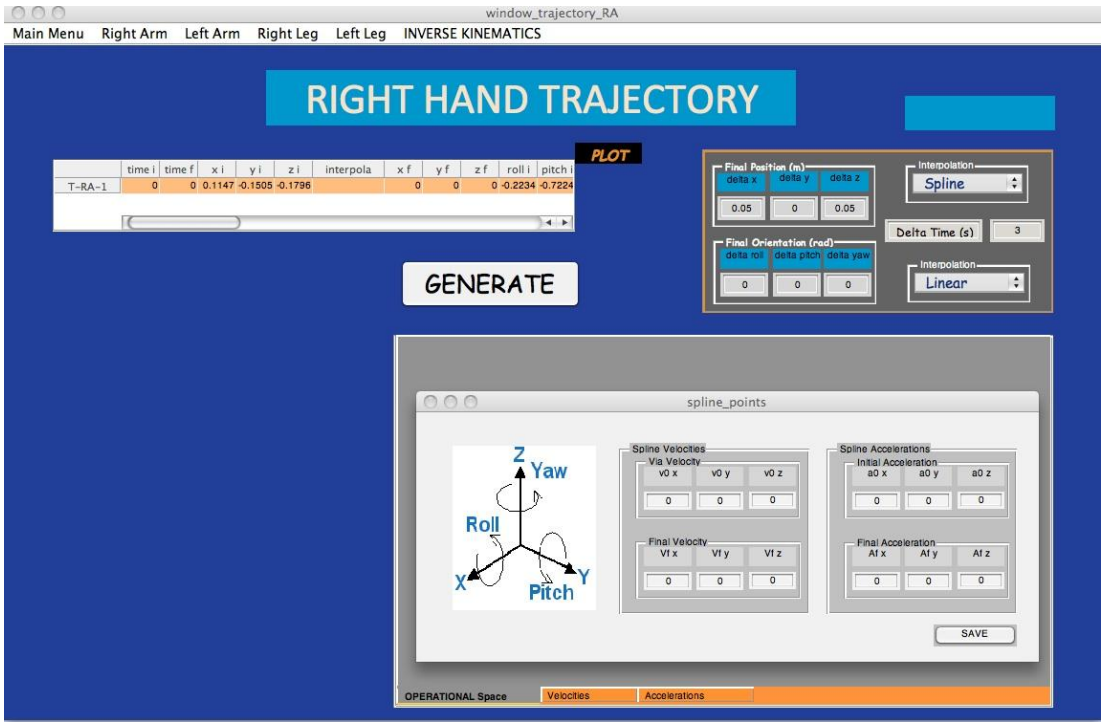



Menú desplegable de selección de trayectorias

En la figura de arriba se observa cómo en los menús de la pierna derecha e izquierda se puede elegir entre trayectoria con pie como soporte o con pie flotante. Por ejemplo si para la trayectoria 1 de la pierna derecha se elige como soporte, la trayectoria flotante quedará bloqueada en la siguiente trayectoria.

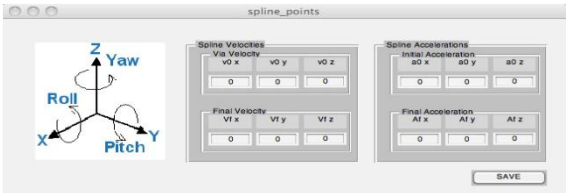
En la parte superior derecha se facilita un panel para que el usuario introduzca la posición y orientación final deseada, el tipo de interpolación a realizar, y la duración de la trayectoria en segundos.



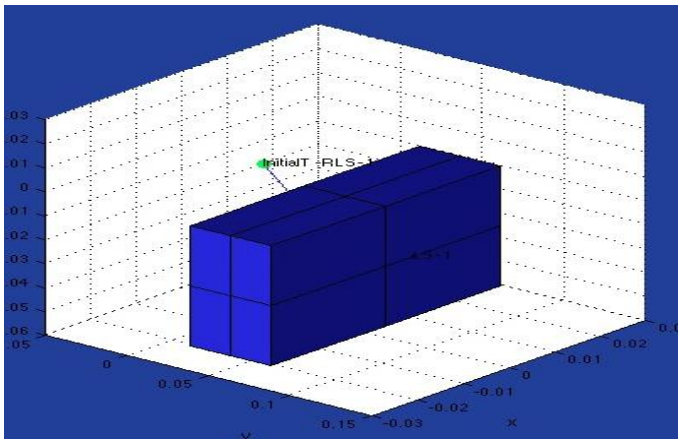

Panel de introducción de posición, orientación, tipo de interpolación y duración de la trayectoria

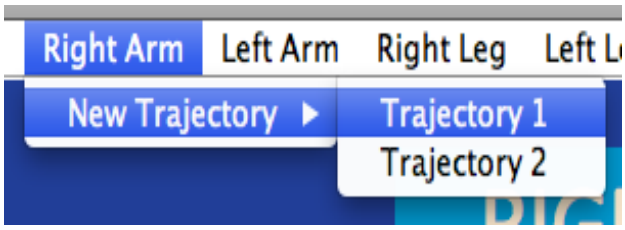

Una vez se pulsa el botón de generación de la trayectoria (GENERATE), dependiendo de el tipo de interpolación elegida, aparecen una ventana de inserción de velocidades y aceleraciones inicial y final.




Una vez introducidos todos los datos, el programa inicia el proceso de la generación de la trayectoria, y una vez finalizado aparece la opción de representar gráficamente dicha evolución temporal de la posición (x,y,z) y la orientación (roll, pitch,yaw) en el espacio operacional así como en el espacio de velocidades y aceleraciones pinchando en la pestaña adecuada.

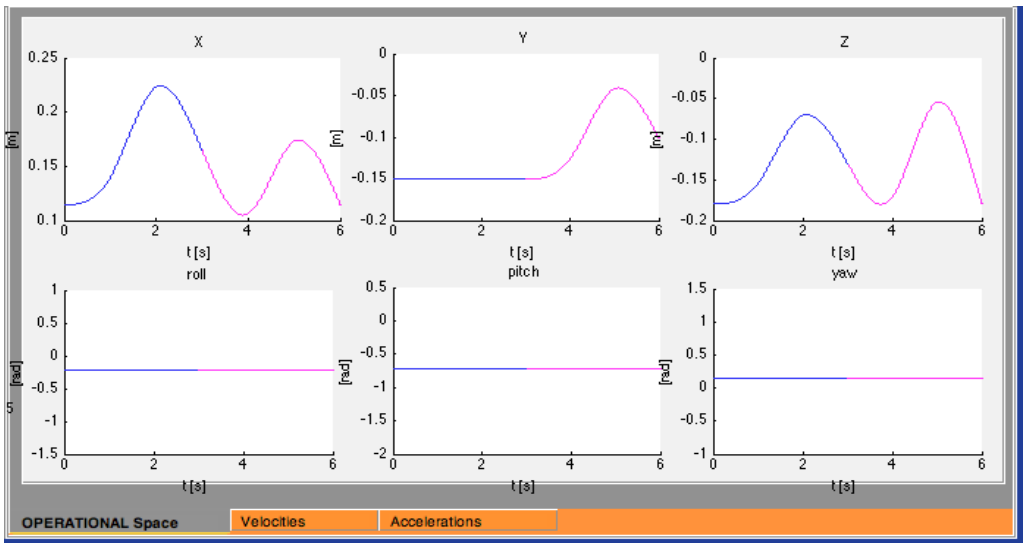
En el display de la zona inferior izquierda de la pantalla, se encuentra la representación gráfica en tres dimensiones del extremo del manipulador elegido.



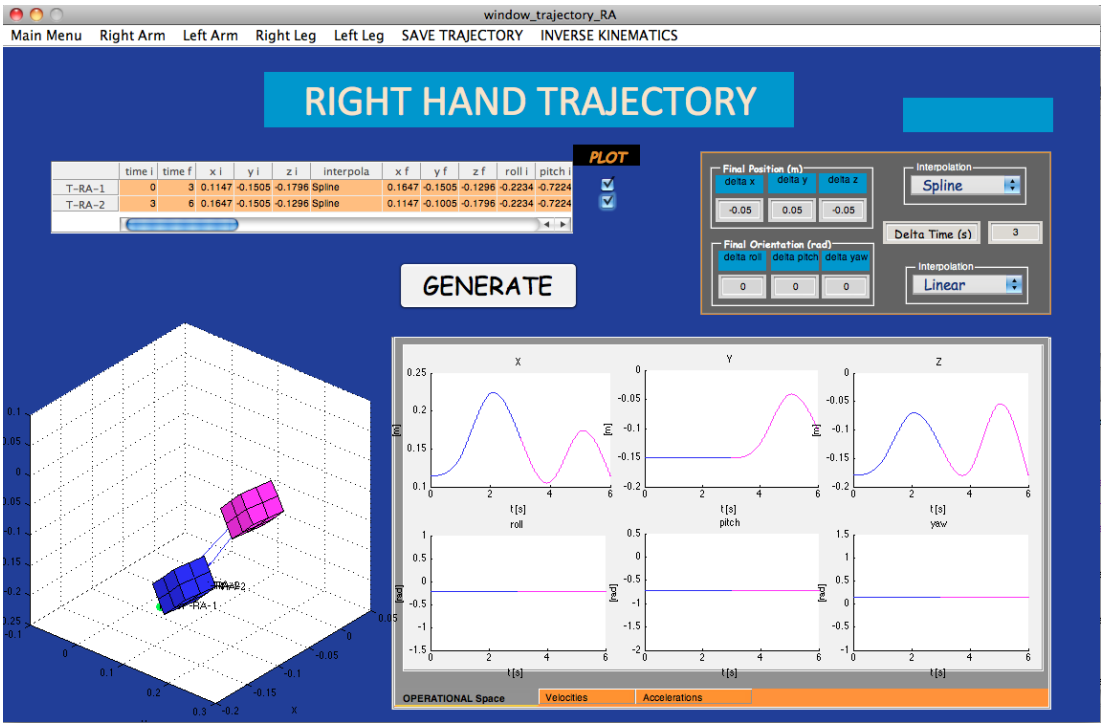



En cada ventana del manipulador podemos generar una trayectoria adicional tomando como punto de partida la posición final de la trayectoria generada en primera instancia sin más pinchar en el menú del manipulador y eligiendo la opción de trayectoria 2 (Trajectory-2).

Si representamos esta nueva trayectoria, aparecerán las curvas anteriormente comentadas con la salvedad de que el primer tramo corresponde a la trayectoria 1 (curva color azul) y de manera continua un segundo tramo correspondiente a la trayectoria 2 (curva color rosa).

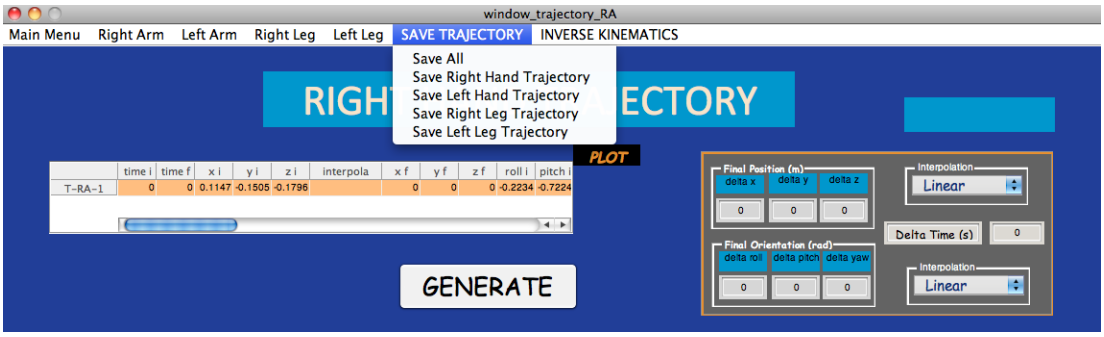



Si deseamos aislar cualquiera de los dos tramos de la trayectoria final, podemos deseleccionar la casilla del check plot para que oculte la trayectoria no deseada.




Ventana del manipulador seleccionado una vez generamos la trayectoria

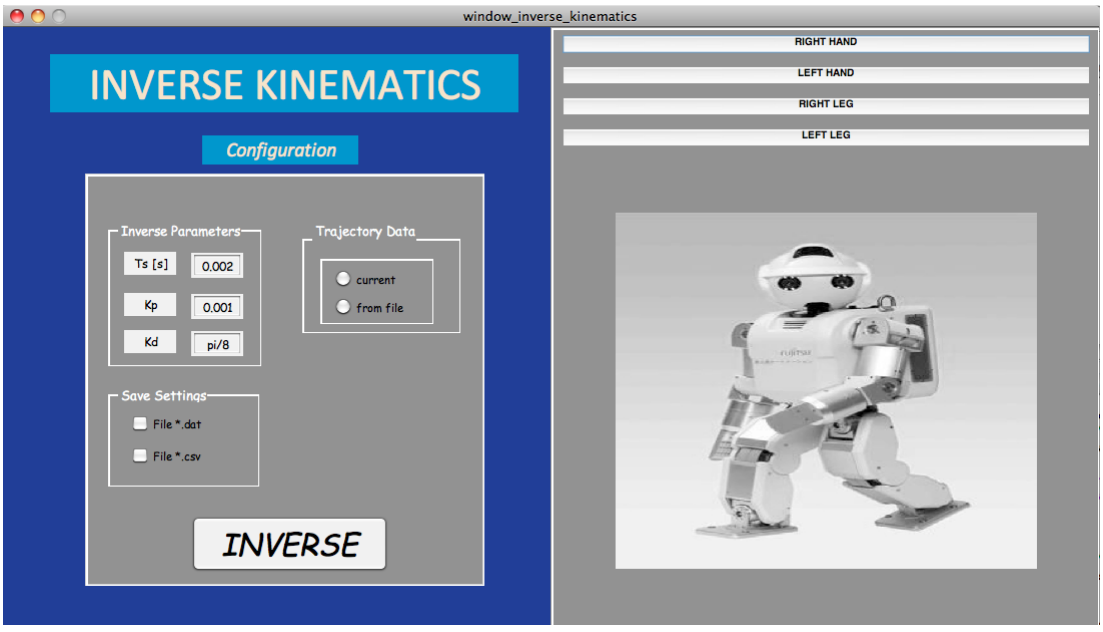
Todas estas trayectorias se podrán salvar en un archivo csv o txt antes de pasar al menú de inversión cinemática.




Menú para salvar las trayectorias generadas de cada manipulador

## Inversión cinemática

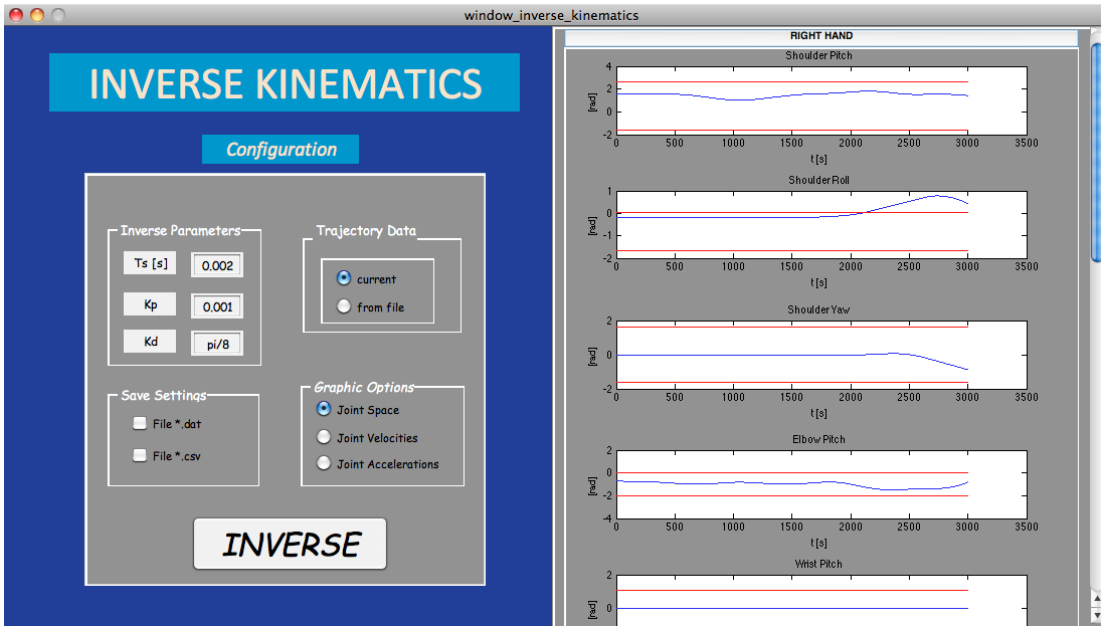
En el menú que nos ocupa, el usuario puede modificar los parámetros del algoritmo de inversión pero en este caso no se puede modificiar el paso bien porque ya se han generado las trayectorias en el menú anterior, bien porque el usuario decide realizar la inversión cinemática a partir de un archivo csv ó txt que almacena una trayectoria anterior.




Aspecto del subprograma de inversión cinemática

Pulsando el botón de inversión (INVERSE) aparece una barra de espera mientras el programa realiza las operaciones de inversión. Una vez finalizado el proceso, se puede elegir una representación del espacio articular de cada

manipulador, de sus velocidades o de sus aceleraciones en el panel Graphic Options.




Vista del panel de representación de características del manipulador (Graphic Options)

Estas gráficas vienen recogidas en las pestañas pop-up situadas en la parte derecha de la pantalla para cada manipulador.

El principal interés de estas gráficas radica en la verificación de que nos encontramos dentro de los límites de giro de cada motor. Si las líneas representadas en la gráfica sobrepasan esos límites, indica que la trayectoria generada no puede ser realizada por la limitación operativa de los motores.

# Steps Control

## Introducción al control de paso

En este segundo bloque de la interfaz se va a generar un paso del robot. Cabe destacar que la generación del paso se hace en base a criterios de estabilidad estática del centro de masas, en ningún momento se tienen en cuenta criterios de la dinámica del robot.

Esta vez, el usuario tiene un control más restringido de los movimientos de los manipuladores, debido a las peculiares características que el movimiento de paso posee.

## Definición de los parámetros de paso

En este apartado vamos a distiguir dos tipos de parámetros de paso. La vista de la pantalla inicial de este bloque de la interfaz se adjunta a continuación:

Aspecto de la pantalla incial del bloque STEP CONTROL de la interfaz

Tipos de soporte del paso:

**- Doble soporte:** es el movimeinto del robot en el que ambos pies se encuentran fijos y que sirve de antesala del inicio del movimiento de simple apoyo.

Figura representativa del movimiento incial de doble soporte

**- Simple soporte:** es aquel movimiento del robot manteniendo uno de los pies apoyado.

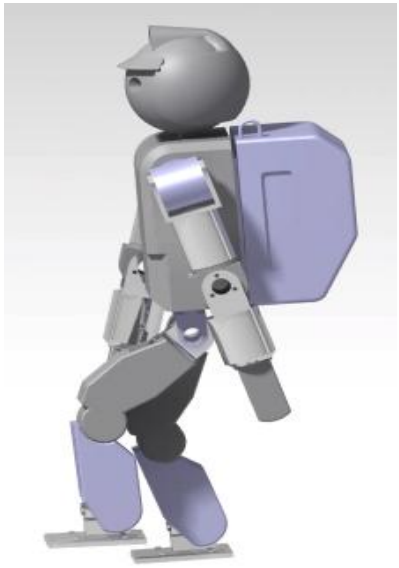
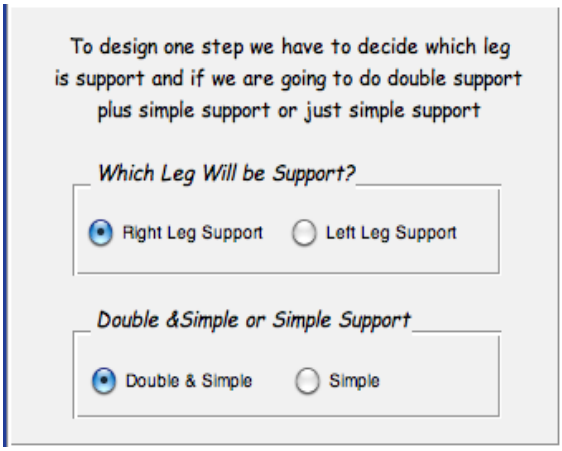
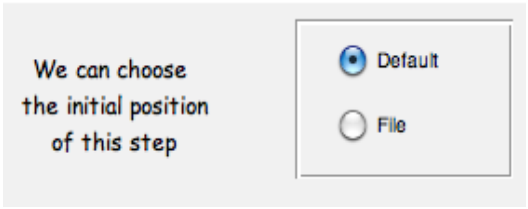


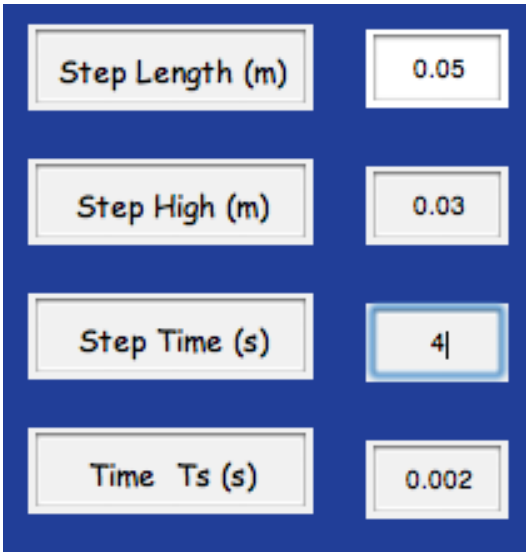
Figura representativa del movimiento inicial de simple soporte

El usuario podrá elegir qué pie hace de soporte, si el derecho o el izquierdo. Además tendrá la opción de hacer doble y simple soporte o solamente simple soporte.




Al igual que en la generación de trayectorias por manipuladores, el usuario puede elegir una posición inicial desde un archivo csv o una por defecto proporcionada por el programa.



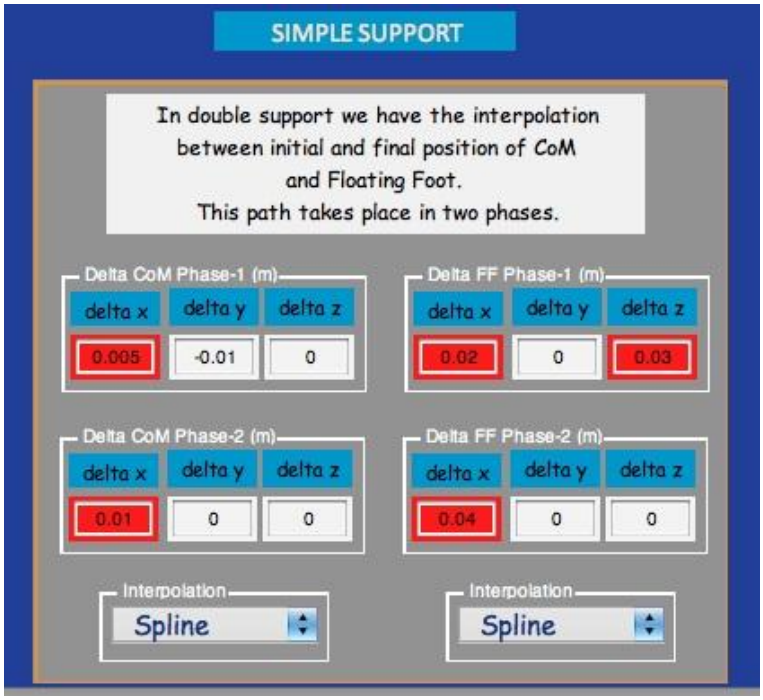

Las variables a introducir son:

- **Longitud del paso:** por defecto esta distancia se reparte por igual entre la aportación del movimiento del centro de masas y del desplazamiento del pie flotante.
- **Altura del paso:** la altura de paso se considera el punto final de la fase 1 de simple soporte y el punto de inicio de la fase 2 de simple soporte cuyo punto final es el suelo.
- **Duración del movimiento total**
- **Tiempo de paso Ts**

## Diseño del paso

Con el fin de clarificar las opciones que la interfaz ofrece, el usuario dispone de notas aclaratorias en la parte superior de cada panel.

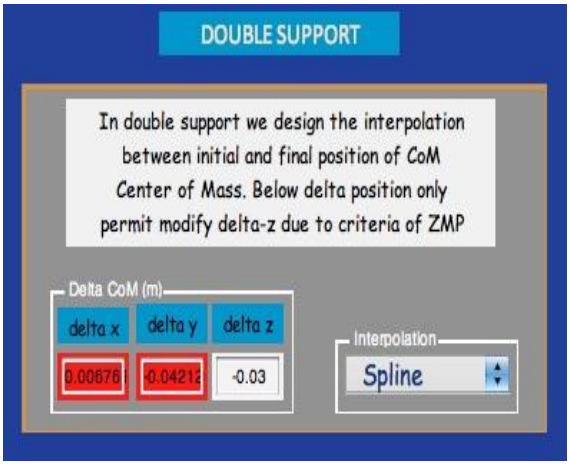
En esta ventana podemos encontrar un panel asociado a cada movimiento a estudiar, es decir, simple soporte, doble soporte y movimiento de brazos.




Panel de introducción de datos del movimiento de simple soporte

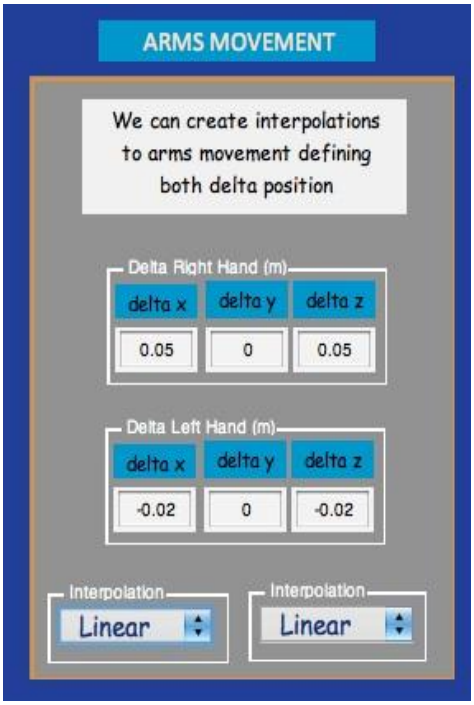
Las celdas en rojo no podrán ser modificas por el usuario debido a que están basadas en las mínimas restricciones de estabilidad que el robot precisa para realizar un paso en condiciones de seguridad.

En cada uno de estos movimientos se tiene la posibilidad de elegir el tipo de interpolación para cada movimiento, pudiéndose combinar interpolaciones distintas en un mismo panel, por ejemplo en el panel de simple soporte, se puede usar una interpolación lineal para la aportación en el movimiento del centro de masas y una interpolación spline para la parte de movimiento correspondiente al pie flotante.



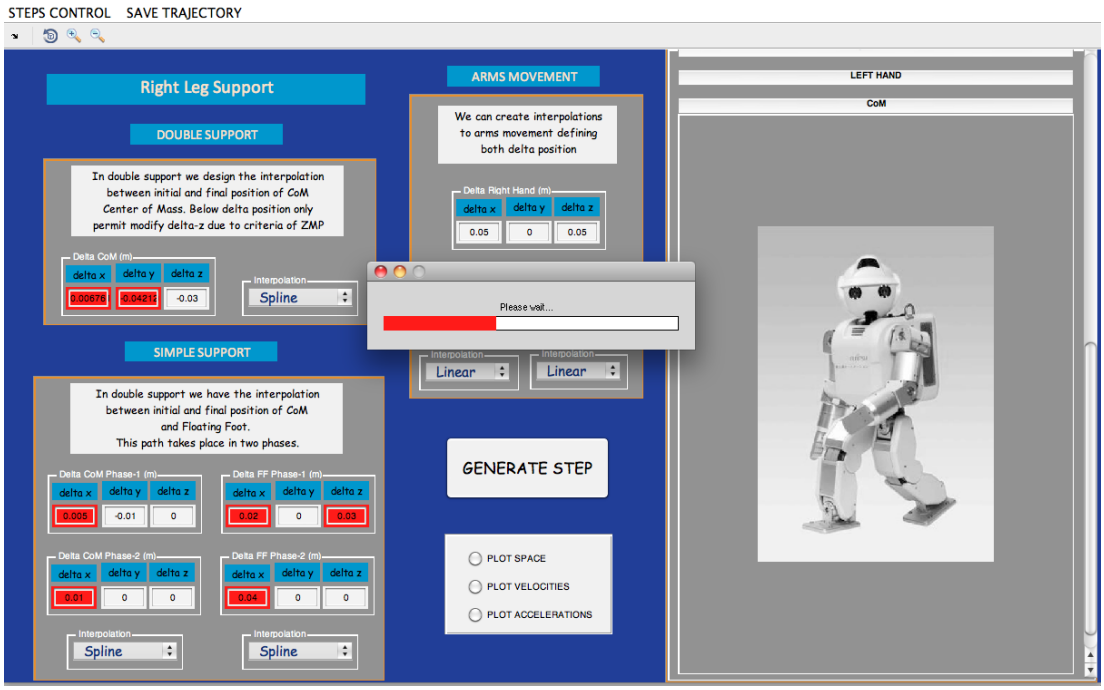

Panel de introducción de datos del movimiento de doble soporte

Notar que en el caso de doble soporte no se va a poder combinar entre dos tipos de interpolaciones distintas porque solo se genera la trayectoria de un punto, el centro de masas.

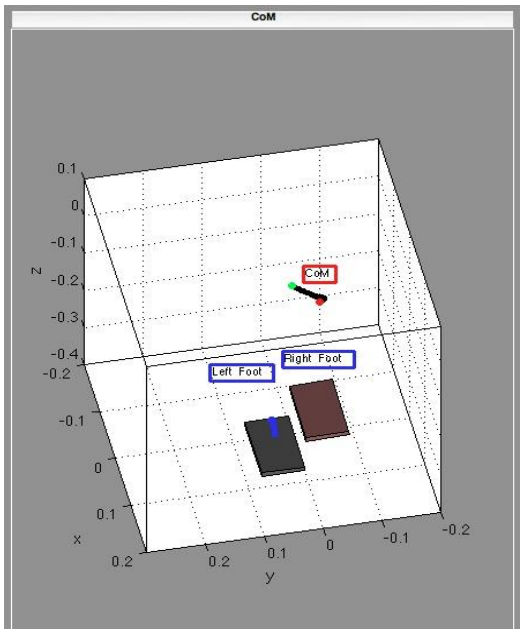



Panel de introducción de datos del movimiento de brazos

Una termine la computación del paso generado tras pulsar el botón GENERATE STEP, automáticamente aparece una simulación en tres dimensiones del movimiento de ambos pies y el centro de masas en la pestaña CoM (Center of Mass)

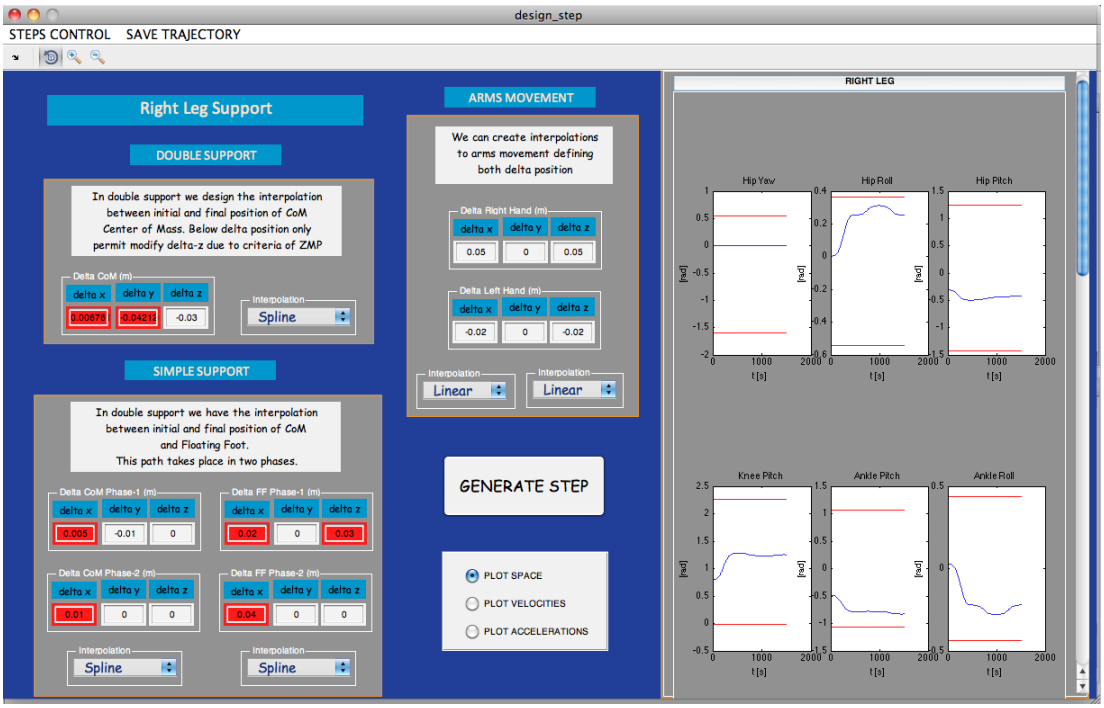



Pantalla de computación del movimiento de paso seleccionado



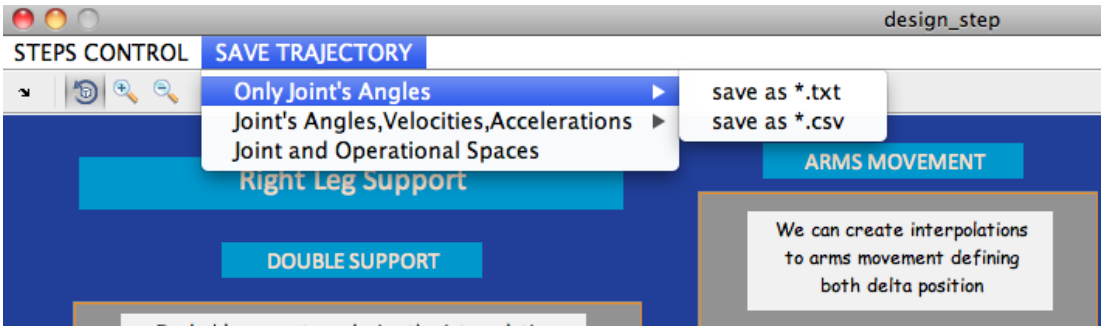

Simulación en tres dimensiones del paso generado

De nuevo se da la posibilidad de representar gráficamente el espacio de articulaciones, las velocidades o las aceleraciones, tal y como se hacía en Manipulator Control.

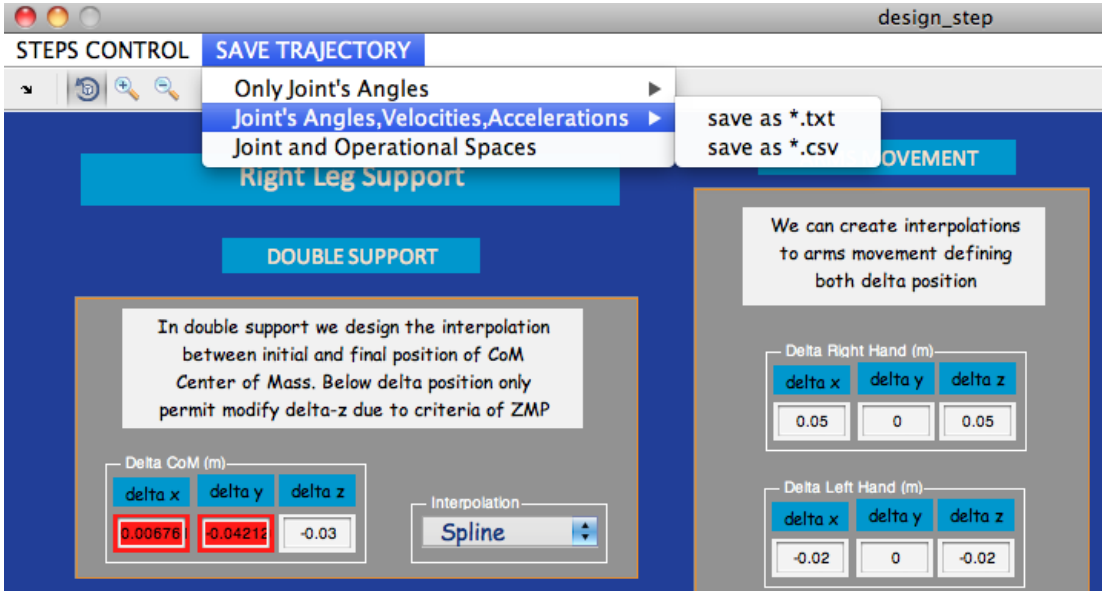



Representación del espacio de articulaciones, velocidad y aceleración

Por último, en el menú SAVE TRAJECTORY se puede grabar los resultados de la inversión cinemática en archivos *.txt o *.csv para un posterior uso en software de simulación, en excel o en Matlab. También cabe la posibilidad de almacenar todos los resultados en una variable *.mat de tipo estructura de datos para uso interno del software Matlab.




Menú SAVE del bloque STEP CONTROL




Menú SAVE del bloque STEP CONTROL